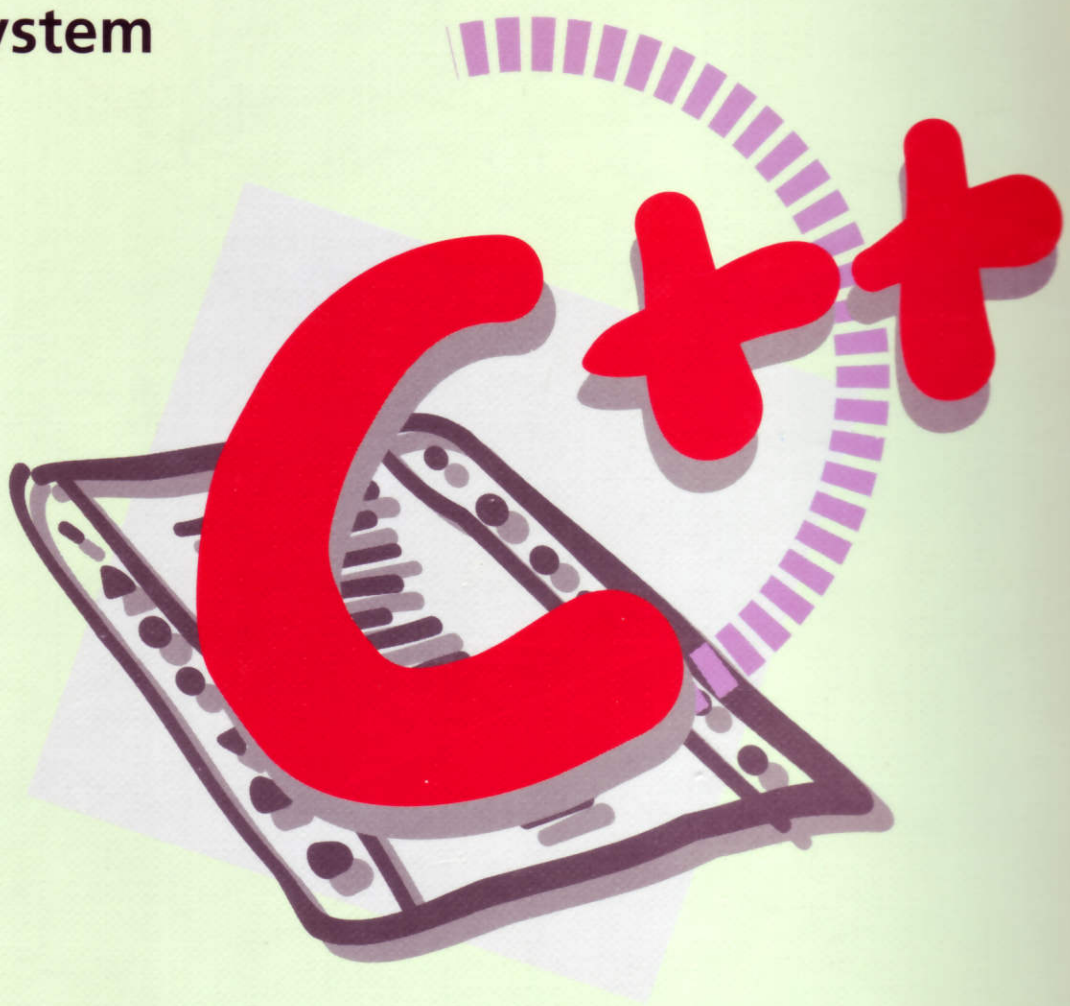
Maxon C++ 3

Integriertes Compilersystem



AMIGA®

MAXON computer

AMIGA

MaxonC++

68000/10/20/30/881/882/851 Assembler

MAXON
computer

MaxonASM 731

1. Einleitung 733

2. Grundlagen der Assemblerprogrammierung 734

2.1 Warum Assembler? 734

2.2 Der Aufbau eines Quelltextes 734

3. Der Assembler 735

3.1 Die Adressierungsarten des MC 68000 735

3.2 Symbole und mathematische Ausdrücke 737

3.2.1 Globale Symbole 737
3.2.2 Lokale Symbole 738
3.2.3 Label 738
3.2.4 Variablen 738
3.2.5 Reservierte Symbole 739
3.2.5.1 NARG 739
3.2.5.2 __RS 739
3.2.5.3 _MOVEMBYTES 739
3.2.6 Mathematische Ausdrücke 740
3.2.7 Zahlenformate 740
3.2.8 Operationen und deren Prioritäten 740

3.3 Macros 741

Schlüsselwörter/Funktionen 743
1. *VALOF(<Symbol>) 743
2. *STRLEN(<String>) 744
3. *LEFT(<String>,n) 744
4. *RIGHT(<String>,n) 744
5. *MID(<String>,s,n) 744
6. *UPPER(<String>) 745

3.4 Bedingte Assemblierung 746

3.5 ALIGNment 747

3.6 Sektionierung 748

Direktiven zur Steuerung der Code-Erzeugung 749

3.7 Erzeugung von absolutem Code 750

3.8 Konstanten 751
3.9 Reservierte Speicherblöcke 753
3.10 Definition einer Variablen 753
3.11 Definition von Strukturoffsets 754
3.12 Externe Quelltexte 754
3.13 Externe Rohdateien 755
3.14 Copper-Assembler 755
3.15 Externe Symbole 758
3.16 Codearten 760
3.17 Assemblerbetriebsarten 760
3.18 Mögliche Optimierungen 761
3.19 Ausgabedateien 763

Anhang A 765
Die Befehle des MC 68000 / MC 68010 765
Die zusätzlichen Befehle des MC 68010 768

Anhang B 769
Die Assemblerdirektiven 769

Anhang C: Die Fehlermeldungen 773
DOS-Fehler beim Dateihandling 773
Fehler in der Sektionierung 773
MACRO-Fehler 773
Fehler beim Parsen einer Zeile 775
Fehler bei der Labeldefinition 775
Fehler im math. Ausdruck 775
Fehler im Addressmode 776
Fehler im ersten Befehlsoperanden 777
Fehler im zweiten Befehls-Operanden 778

Fehler im dritten Befehls-Operanden 778
Fehler in den IF-Konstruktionen 778
Spezielle Fehler bei Branch-Befehlen 779
Spezielle Fehler bei EQU 779
Spezielle Fehler bei EQUR 779
Spezielle Fehler bei REG 779
Spezielle Fehler bei SET 780
Spezielle Fehler bei CNOP/ALIGN 780
Spezielle Fehler bei XDEF 780
Spezielle Fehler bei XREF 780
Spezielle Fehler bei DC.x 780
Spezielle Fehler bei RS.x 780
Spezielle Fehler bei INCBIN 781
Spezielle Fehler bei BASEREG 781
Allgemeine Fehler 781
Interne Assembler-Fehler 781
Fehler bei FPU-Operationen 782
Fehler im Addressmode 782
Fehler im zweiten Befehlsoperanden 782
Fehler im Addressmode 782
Fehler im ersten Operanden 783
Fehler im zweiten Operanden 783
Fehler bei Copper-Operationen 783

Anhang D: Register 784

Anhang E, Befehle 785

1. Einleitung

MaxonASM-CLI ist eine spezielle Version des integrierten Komplettpakets MaxonASM. Es handelt sich dabei nicht um eine abgespeckte Fassung, so daß alle Features vorhanden sind. Der Assembler unterstützt alle Codes der Prozessoren 68000/10/20/30, der FPUs 68881/2 sowie der PMMUs 68851/68030. Desweiteren können Makros und Includes genutzt werden. Die Assembliergeschwindigkeit ist beim MaxonASM sehr hoch, je nach Programm erreicht er schon auf einem 68000-Amiga Geschwindigkeiten von 60000-90000 Zeilen pro Minute.

Sie erhalten mit der CLI-Version des MaxonASM also einen Assembler, den Sie neben der Verwendung mit MaxonC^{++} auch für größere Assembler-Projekte komfortabel einsetzen können.

2. Grundlagen der Assemblerprogrammierung

Eine komplette Einführung in die Assembler-Programmierung würde den Rahmen dieses Handbuches bei weitem sprengen. Bitte verstehen Sie deshalb dieses Kapitel nur als Überblick über die Möglichkeiten, die Ihnen Assembler bietet.

2.1 Warum Assembler?

Schon seit geraumer Zeit gewinnen Hochsprachen immer mehr an Bedeutung. Ein Beispiel für den Trend hin zu Hochsprachen ist das Betriebssystem des Amiga. Es wurde zum größten Teil in C, DOS sogar in BCPL verfaßt (ab OS 2.0 ist DOS in C geschrieben). Der Vorteil von Hochsprachen liegt in der einfacheren Programmierung und der geringeren Fehleranfälligkeit. Im allgemeinen sind auch Hochsprachenprogramme leichter zu lesen und zu durchschauen. Aber auch die Nachteile von Hochsprachen sollen nicht verschwiegen werden. Die Programme müssen erst von einem Compiler in eine für den Prozessor verständliche Form (Maschinensprache) übersetzt werden. Der enstehende Code ist nicht besonders schnell und meist länger als der direkt in Assembler geschriebener Programme. Assembler bietet zudem noch den Vorteil, sich nicht mit vielen verschiedenen Variablentypen auseinandersetzen zu müssen.

2.2 Der Aufbau eines Quelltextes

Die Zeile eines Quelltextes gliedert sich grob in 4 Teile:

Labelfeld	Opcodefeld	Operandenfeld	Kommentar
`Start`	`LEA`	`Label,A1`	`;Beispiel`

Ganz links, direkt in der ersten Spalte, stehen Label. Label sind Sprungmarken. Diese Sprungmarken haben die Aufgabe, die von Basic her bekannten Zeilennummern zu ersetzen und das Programm lageunabhängig assemblieren zu können. Sie dürfen für Label beliebig lange Namen verwenden. Durch mindestens ein Leerzeichen vom Rand bzw. Label getrennt beginnt das Opcode-Feld. Hier stehen die Befehle (Mnemonics) bzw. Direktiven für den Assembler. Auf den Befehl folgen die Operanden. Mehrere Operanden werden durch Kommata getrennt. Zum Schluß der Zeile können Sie noch einen Kommentar anfügen. Kommentare werden mit einem Semikolon bzw. mit einem „*“ einge- leitet. Sie können natürlich einzelne Teile einer Zeile weglassen, z.B. ist auch

```
Start       ; Zeile ohne Opcode
```

oder

```
moveq   #0,d0    ; Zeile ohne Label
```

oder

```
; Zeile ohne Label und Opcode
```

richtig.

Sie sollten Ihren Quelltext mit möglichst vielen Kommentaren ausstatten, damit Sie sich auch nach längerer Zeit noch in dem entsprechenden Programm zurechtfinden. Sehr hilfreich ist es, den Quelltext mit einem sogenannten Kommentarkopf zu versehen, in dem der Name des Programms, eine grobe Beschreibung der Funktion, die Versionsnummer und eventuell auch Copyright-Vermerke stehen.

3. Der Assembler

Er unterstützt die Programmierung der Prozessoren 68000,68010,68020 und 68030, der FPUs 68881/68882 und der MMU 68851. Selbstverständlich sind auch die Befehle zur Programmierung der im MC 68030 integrierten MMU implementiert.

Den externen Assembler rufen Sie wie folgt auf:

```
asm  [FROM] [PROG] <Quelldatei> [-o <Objektdatei>]
     [-l <Listingdatei>] [-v <Fehlerdatei>]
     [-t <Symboltabelle>] [-c <Optionen>]
```

folgende Optionen sind dabei möglich :

- **c** - es wird zwischen Groß- und Kleinschreibung unterschieden
- **w** - Warnungen werden unterdrückt
- **o** - aktiviert den Multipass-Optimierer
- **s** - es werden Symbolhunks erzeugt
- **a** - es wird absoluter Code erzeugt, die Datei ist nicht linkbar bzw. vom CLI aus ladbar !
- **e** - es gelangen nur exportierte Symbole in den Symbolhunk
- **m** - Macros werden in Listing expandiert
- **l** - es wird eine linkbare Datei erzeugt

3.1 Die Adressierungsarten des MC 68000

Der MC 68000 ist ein 16/32-Bit-Prozessor. 16 Bit deshalb, weil sein Datenbus nur 16 Bit breit ist, und 32 Bit, weil die Operationen innerhalb des Prozessors bis auf wenige Ausnahmen mit einer Breite von 32 Bit ablaufen.

Der MC 68000 verfügt über 8 Daten- und 8 Adressregister, von denen allerdings nur die ersten 7 frei zur Verfügung stehen, da das 8. Adressregister als Stackpointer benutzt wird. Die Datenregister erhalten die Bezeichnungen D0 bis D7 und die Adressregister A0 bis A7. Sie können für A7 auch die Bezeichnung SP verwenden. Weiterhin gibt es ein sogenanntes Statusregister, in dem die für bedingte Verzweigungen wichtigen Flags und im oberen Byte wichtige Bits, die den Zustand des Prozessors beschreiben, liegen. Das gesamte Register (16 Bit) erreichen Sie mit SR, das unter Byte, das die sogenannten Conditioncodes enthält, heißt CCR. Für die zwei Betriebsmodi des MC 68000, den normalen Usermode und den Supervisormode, existieren noch zwei Stackpointer zur Verwal-

tung getrennter Stackbereiche. Zum einen ist das der USP für den Usermode und zum anderen der SSP für den Supervisiormodus. Der Zugriff auf diese beiden Register sowie auf das SR (schreibend) ist privilegiert, d.h. nur im Supervisormodus erlaubt.

Adressierungsarten

Adressierungsart	Syntax alt	Syntax neu	000	010	020	030
Datenregister direkt	Dn	Dn	*	*	*	*
Adreßregister direkt (AR)	An	An	*	*	*	*
AR indirekt (ARI)	(An)	(An)	*	*	*	*
ARI mit Postinkrement	(An)+	(An)+	*	*	*	*
ARI mit Predekrement	-(An)	-(An)	*	*	*	*
ARI mit 16 Bit Distanz	d16(An)	(d16,An)	*	*	*	*
ARI mit 8 Bit Dist.,Index	d8(An,Rn.x)	(d8,An,Rn.x)	*	*	*	*
ARI,d8,Index,Scalar	d8(An,Rn.x*sc)	(d8,An,Rn.x*sc)			*	*
ARI,BD,Index	bd(An)	(bd,An)			*	*
	(Rn.x)	(Rn.x)			*	*
	bd(Rn.x)	(bd,Rn.x)			*	*
	bd(An,Rn.x)	(bd,An,Rn.x)			*	*
ARI,BD,Index,Scalar	(Rn.x*sc)	(Rn.x*sc)			*	*
	bd(Rn.x*sc)	(bd,Rn.x*sc)			*	*
	bd(An,Rn.x*sc)	(bd,An,Rn.x*sc)			*	*
Speicher indirekt Postindex	([bd])	([bd])			*	*
	([An])	([An])			*	*
	([bd,An])	([bd,An])			*	*
	([bd],Rn.x)	([bd],Rn.x)			*	*
	([bd],Rn.x*sc)	([bd],Rn.x*sc)			*	*
	([An],Rn.x)	([An],Rn.x)			*	*
	([An],Rn.x*sc)	([An],Rn.x*sc)			*	*
	([bd,An],Rn.x)	([bd,An],Rn.x)			*	*
	([bd,An],Rn.x*sc)	([bd,An],Rn.x*sc)			*	*
	([bd],od)	([bd],od			*	*
	([An],od)	([An],od)			*	*
	([bd,An],od)	([bd,An],od)			*	*
	([bd],Rn.x,od)	([bd],Rn.x,od)			*	*
	([bd],Rn.x*sc,od)	([bd],Rn.x*sc,od)			*	*
	([An],Rn.x,od)	([An],Rn.x,od)			*	*
	([An],Rn.x*sc,od)	([An],Rn.x*sc,od)			*	*
	([bd,An],Rn.x,od)	([bd,An],Rn.x,od)			*	*
	([bd,An],Rn.x*sc,od)	([bd,An],Rn.x*sc,od)			*	*
Speicher indirekt Preindex	([bd,Rn.x])	([bd,Rn.x])			*	*
	([bd,Rn.x*sc])	([bd,Rn.x*sc])			*	*
	([An,Rn.x])	([An,Rn.x])			*	*
	([An,Rn.x*sc])	([An,Rn.x*sc])			*	*
	([bd,An,Rn.x])	([bd,An,Rn.x])			*	*
	([bd,An,Rn.x*sc])	([bd,An,Rn.x*sc])			*	*
	([bd,Rn.x],od)	([bd,Rn.x],od)			*	*
	([bd,Rn.x*sc],od)	([bd,Rn.x*sc],od)			*	*
	([An,Rn.x],od)	([An,Rn.x],od)			*	*
	([An,Rn.x*sc],od)	([An,Rn.x*sc],od)			*	*
	([bd,An,Rn.x],od)	([bd,An,Rn.x],od)			*	*
	([bd,An,Rn.x*sc],od)	([bd,An,Rn.x*sc],od)			*	*
Absolut kurz	$xxx.w	($xxx).w	*	*	*	*
Absolut lang	$xxx.l	($xxx).l	*	*	*	*
PC-rel. mit 16 Bit Distanz	d16(PC)	(d16,PC)	*	*	*	*
PC-rel. mit 8 Bit Dist.,Index	d8(PC,Rn.x)	(d8,PC,Rn.x)	*	*	*	*
PC-rel. d8,Index,Scalar	d8(PC,Rn.x*sc)	(d8,PC,Rn.x*sc)			*	*

```
PC-rel., BD,Index                 bd(PC)            (bd,PC)                  *   *
                                bd(PC,Rn.x)       (bd,PC,Rn.x)               *   *
PC-rel., BD,Index,Scalar       bd(PC,Rn.x*sc)    (bd,PC,Rn.x*sc)             *   *
PC-rel.,Speicher ind. Postinx.  ([PC])              ([PC])                   *   *
                                 ([bd,PC])          ([bd,PC])                *   *
                                ([PC],Rn.x)        ([PC],Rn.x)               *   *
                               ([PC],Rn.x*sc)     ([PC],Rn.x*sc)             *   *
                               ([bd,PC],Rn.x)     ([bd,PC],Rn.x)             *   *
                              ([bd,PC],Rn.x*sc)  ([bd,PC],Rn.x*sc)           *   *
                               ([bd,PC],od)       ([bd,PC],od)              *   *
                              ([PC],Rn.x,od)     ([PC],Rn.x,od)             *   *
                             ([PC],Rn.x*sc,od)  ([PC],Rn.x*sc,od)           *   *
                             ([bd,PC],Rn.x,od)  ([bd,PC],Rn.x,od)           *   *
                            ([bd,PC],Rn.x*sc,od) ([bd,PC],Rn.x*sc,od)       *   *
PC-rel.,Speicher ind. Preinx. ([PC,Rn.x])        ([PC,Rn.x])                *   *
                               ([PC,Rn.x*sc])     ([PC,Rn.x*sc])             *   *
                               ([bd,PC,Rn.x])     ([bd,PC,Rn.x])             *   *
                              ([bd,PC,Rn.x*sc])  ([bd,PC,Rn.x*sc])           *   *
                              ([PC,Rn.x],od)     ([PC,Rn.x],od)             *   *
                             ([PC,Rn.x*sc],od)  ([PC,Rn.x*sc],od)           *   *
                             ([bd,PC,Rn.x],od)  ([bd,PC,Rn.x],od)           *   *
                            ([bd,PC,Rn.x*sc],od) ([bd,PC,Rn.x*sc],od)       *   *
Konstante unmittelbar            #k                  #k        *   *          *   *
```

Abkürzungen :

`Dn`	Dateneregister `D0...D7`
`An`	Adreßregister `A0...A7 (SP)`
`Rn.x`	Register `D0...D7` oder `A0...A7 (SP)` und optionale Länge `.W` oder `.L`
`sc`	Skalar, mit dem Indexregister multipliziert wird. Erlaubte Werte : 1,2,4 oder 8
`d16`	16-Bit-Distanz
`d8`	8-Bit-Distanz
`bd`	Base Displacement 16 oder 32 Bit
`od`	Outer Displacement 16 oder 32 Bit

Der beim MC 68000 nicht privilegierte Befehl `MOVE SR,<ea>` darf ab dem MC 68010 nur noch im Supervisor-Mode benutzt werden !

3.2 Symbole und mathematische Ausdrücke

Bei den Symbolen ist grundsätzlich zwischen globalen und lokalen zu unterscheiden. Globale Symbole gelten im gesamten Quelltext während lokale Symbole nur zwischen zwei globalen Symbolen bzw. innerhalb eines Macros gelten.

3.2.1 Globale Symbole

Globale Symbole beginnen immer miteinem Buchstaben bzw. einem Underscore (`"_"`). Danach dürfen weitere Buchstaben, Ziffern bzw. Underscores folgen. Alle Zeichen sind signifikant, d.h. die

Namen dürfen beliebig lang sein und alle Zeichen werden verglichen. Es wird grundsätzlich nicht zwischen Groß- und Kleinschreibung unterschieden.

```
DosBase dc.l    0    ;DosBase ist ein globales Label
LN_SUCC RS.L    1    ;LN_SUCC ist eine globale Variable
```

3.2.2 Lokale Symbole

Lokale Symbole gelten solange, bis wieder ein globales Symbol definiert wird. Zusätzlich gibt es einen Sonderfall, es gibt auch lokale Symbole, deren Gültigkeit auf ein Macro beschränkt ist.
Symbole gelten als lokal, wenn sie:

1. mit einer Ziffer beginnen und mit dem "$"-Zeichen enden:

```
0$  clr.b (a0)+
  dbra d0,0$       ;0$ ist ein lokales Label
```

2. mit einem Punkt beginnen:

```
.Loop    clr.b (a0)+
         dbra d0,.Loop
```

3. mit einem Backslash ("\") beginnen:

```
\Loop    clr.b (a0)+
         dbra d0,\Loop
```

Diese Möglichkeit sollte allerdings innerhalb von Macros vermieden werden, da sie zu Kollisionen mit den Parameterbezeichnungen führt.

3.2.3 Label

Als Label bezeichnet man Sprungmarken, die den Stand des PC an der Stelle, an der sie definiert wurden, repräsentieren. Sie dienen als Ziel für Verzweigungen bzw. absoluter oder PC-relativer Adressierungen. Ein Label wird definiert, wenn es in einer Zeile ganz links beginnt und keine Direktive wie **EQU**, **SET** bzw. **RS** folgt. Label dürfen sowohl lokal als auch global sein.

Beispiele:

```
Das_ist_ein_Label
doslib.name      ;auch der Punkt darf vorkommen
doslib.l         ;ist verboten, da .l als Größenangabe gilt!
.Start           ;ein lokales Label
```

3.2.4 Variablen

Variablen bekommen im Gegensatz zu den Labeln mittels einer Direktive einen Wert zugewiesen, der völlig unabhängig vom aktuellen PC ist. Variablen sind nur dann redefinierbar, d.h. man darf ihnen an beliebiger Stelle einen neuen Wert zuweisen, wenn sie mit der Direktive **SET** definiert wurden.

3.2.5 Reservierte Symbole

Es existieren vier reservierte Symbole, deren Namen Sie also nicht für eigene Symbole verwenden dürfen.

3.2.5.1 NARG

Dieses Symbol darf nur innerhalb von Macros verwendet werden und enthält immer die Anzahl der Parameter, die dem Macro übergeben wurden.

z.B.:

```
IFEQ NARG-2
...        ;diese Zeilen werden nur assembliert, wenn dem
...        ;Macro genau zwei Parameter übergeben wurden
ENDIF
```

Ein weiteres Beispiel für die Benutzung von NARG finden Sie im Demoprogramm #1.

3.2.5.2 __RS

Das Symbol **__RS** enthält den aktuellen Stand des RS-Zählers. Durch die Direktiven **RS**, **RSSET** und **RSRESET** wird dieser Zähler verändert.

z.B.:

```
   RSRESET          ; __RS = 0 !
X1 RS.L 1           ; __RS = 4
X2 RS.L 1           ; __RS = 8
Y1 RS.L 1           ; __RS = 12
Y2 RS.L 1           ; __RS = 16
   RSSET 500        ; __RS = 500
Z  EQU __RS         ; Z = 500
```

3.2.5.3 _MOVEMBYTES

Dieses Symbol gibt die Anzahl der mit dem letzten MOVEM-Befehl bewegten Bytes wieder. Dieses Symbol kann man z.B. dazu benutzen, den Stackpointer (**SP**) nach einem **MOVEM**-Befehl zu korrieren ohne die Register wieder vom Stack holen zu müssen:

```
    movem.l d0-d4/d6/a0-a2/a4,-(sp)
    ...
    ...
    beq.s   0$
    add.w   #_MOVEMBYTES,SP      ;Stackpointer korrigieren
    rts
0$  movem.l (sp)+,d0-d4/d6/a0-a2/a4
    rts
```

3.2.6 Mathematische Ausdrücke

Der Assembler berechnet mathematische Ausdrücke unter Berücksichtigung der Priorität der mathematischen Operationen. Sie dürfen auch Klammern benutzen, die maximal 32mal ineinander geschachtelt sein dürfen.

3.2.7 Zahlenformate

Alle Zahlen sind grundsätzlich vom Format Integer 32 Bit(-$80000000 <= n <= $7FFFFFFF bzw. 0 <= n <= $FFFFFFFF)

1. Dezimalzahlen

```
123456
24
065
```

2. Hexadezimalzahlen

```
$FF
$1F
$0080
```

3. Oktalzahlen

```
@70@55@012
```

4. Binärzahlen

```
%10101010
%01100110
```

5. Symbole

```
.Start
_LVOOpenLibrary
__RS
```

3.2.8 Operationen und deren Prioritäten

Pri	Operation		
7	-	negatives Vorzeichen	z.B. `-4, -$F0`
7	~	NOT	z.B. `~1, ~%0010`
6	*	Multiplikation	z.B. `LN_SIZE*4`
6	/	Division	z.B. `640/2`
6	%%	Modulo (Rest aus Division)	z.B. `100%%16`
5	+	Addition	z.B. `Start+4`
5	-	Subtraktion	z.B. `CNT-1`

4	<<	nach links schieben	z.B. `7<<5`
4	>>	nach rechts schieben	z.B. `2>>CNT`
3	&	Logisches UND	z.B. `XY&$7F`
2	^	Logisches EOR	z.B. `XY^$7F`
1	|	Logisches ODER	z.B. `GADGETUP|GADGETDOWN`
1	!	Logisches ODER	z.B. `GADGETUP!GADGETDOWN`

Beispiele:

```
(LN_SIZE>>2)-1
2*(4+5)              ; gibt 18
2*4+5                ; gibt 13
```

3.3 Macros

Macros sind häufig benötigte Befehle oder Befehlsfolgen, die zu einer Einheit zusammengefaßt werden. Ein Macro ist also nichts weiter, als ein neuer Befehl, den Sie definieren und der aus einem oder mehreren Kommandos besteht. Stellen Sie sich vor, Sie wollen die DOS-, Intuition- und Graphics-Library öffnen. Die folgenden Befehle

```
lea Name,a1
moveq   #Lib_Version,d0
jsr _LVOOpenLibrary(a6)
move.l  d0,LibBase
beq Exit
```

müßten Sie also dreimal schreiben. Kürzer und übersichtlicher gestaltet sich dagegen der Einsatz eines Macros:

```
* Macrodefinition

OpenLib     MACRO
lea \1Name,a1
moveq   #Lib_Version,d0
jsr _LVOOpenLibrary(a6)
move.l  d0,\1Base
beq Exit
ENDM
```

Das Macro und somit Ihr neuer Befehl heißt „OpenLib". Das Öffnen aller Libraries sieht dann folgendermaßen aus:

```
* Macroaufruf

OpenLib Dos
OpenLib Int
OpenLib Gfx
...
...
```

```
DosBase     dc.l    0
IntBase     dc.l    0
GfxBase     dc.l    0
DosName     dc.b    "dos.library",0
IntName     dc.b    "intuition.library",0
GfxName     dc.b    "graphics.library",0
```

Trifft der Assembler auf den neuen Befehl „OpenLib", so assembliert er nichts anderes, als die Zeilen, die in der Definition zwischen MACRO und ENDM stehen.

Dem Macro können natürlich auch Parameter übergeben werden. Im obigen Fall übergeben Sie `"Dos"`, `"Int"` bzw. `"Gfx"`. Trifft der Assembler bei der Bearbeitung eines Macros auf die Zeichen `\n`, so ersetzt er diese Zeichenfolge durch den entsprechenden Parameter. n ist die laufende Nummer des Parameters.

Zeile 1 des Macros:

```
lea \1Name,a1
```

wird beim ersten Ausruf also zu:

```
lea DosName,a1
```

Einem Macro dürfen Sie maximal 36 Parameter übergeben, die durch Kommata getrennt werden. Die Parameter werden im Macro mit `\1` bis `\9` und `\A` bis `\Z` bezeichnet. `\A` steht dabei für Parameter Nr. 10 und `\Z` für Nr. 36.

Macros dürfen beliebig lang sein, d.h. die maximale Länge wird nur durch den zur Verfügung stehenden Arbeitsspeicher begrenzt.

Macros dürfen auch ineinander geschachtelt werden, d.h. aus einem Macro heraus kann ein weiteres aufgerufen werden, aus diesem wiederum ein drittes usw. Die maximale Schachteltiefe ist unbegrenzt.

```
CALL    MACRO
jsr _LVO\1(a6)
ENDM

LINKLIB     MACRO
move.l  a6,-(sp)
move.l  \2Base,a6
CALL    \1              ;aus diesem Macro wird ein
                        ;weiteres aufgerufen
move.l  (sp)+,a6
ENDM
```

Eine Macrodefinition beginnt also mit der Direktive **`MACRO`**. In derselben Zeile MUSS der Name des Macros stehen. Alle nun folgenden Zeilen werden in die Macrodefinition übernommen, bis die Direktive **`ENDM`** erreicht wird.

In Macros dürfen natürlich auch Variablen definiert werden. Dabei spielt es keine Rolle, ob es sich um globale oder lokale Variablen handelt. Es kommt jedoch zu Problemen, wenn in einem Macro eine globale Variable definiert wird und Sie dieses Macro mehrmals benutzen. In diesem Fall würde der Assembler versuchen, die Variable mehrfach zu definieren, was insbesondere bei Labeln zu einer Fehlermeldung führen würde. Das gleiche Problem tritt auf, wenn Macros mit Definitionen lokaler Variablen mehrfach aufgerufen werden, ohne daß eine globale Variable definiert wird. Aus diesem Grund gibt es Variablen, die nur innerhalb eines Macros gelten (siehe auch 3.2.2). Sie beginnen mit einem Buchstaben oder Underscore (`"_"`) und enden mit der Kennung `"\@"`.

```
ABS     MACRO
        tst.l   \1
        bpl.s   NotNeg\@
        neg.l   \1
NotNeg\@
        ENDM
```

Dieses Macro ermittelt den Betrag einer Zahl. Das Label `NotNeg\@` gilt nur innerhalb des Macros.

Mit der Assembler-Version 5.0 stehen eine Reihe neuer Macrofunktionen zur Verfügung.

`\@` wird durch „_nnn" ersetzt, wobei nnn die laufende Nummer des Macros ist.
z.B. wird `Loop\@` zu `Loop_4`, wenn der interne Macrozähler auf 4 stand. Die laufende Nummer wird dezimal eingetragen.

`\#` wirkt wie `\@`, nur wird vor nnn kein `"_"` gesetzt. z.B.: aus `0\#$` wird `04$`

`\0-` enthält die Operandenlänge, die beim Macroaufruf benutzt wurde :

```
Test  MACRO
   tst.\0   \1
   ENDM
```

`Test d0` wird zu: `tst.w d0`

`Test.b (sp)` wird zu : `tst.b (sp)`

Schlüsselwörter/Funktionen

Alle folgenden Funktionen werden mit `"\*"` eingeleitet. Das Argument muß in Klammern stehen. Im Argument dürfen weitere Funktionen vorkommen, es darf maximal 16-fach geschachtelt werden.

*1. *VALOF(<Symbol>)*

Die Funktion berechnet den numerischen Wert ihres Argumentes (Symbol) und trägt diesen Wert als Dezimalzahl in die Macrozeile ein:

```
Exec = 4

Test  MACRO
   move.l   \*VALOF(Exec).w,a6
  ENDM
```

```
Test
```
wird zu:

```
move.l 4.w,a6
```

2. *STRLEN(<String>)

Diese Funktion berechnet die Anzahl der Zeichen des Argumentes und trägt die Länge als Dezimalzahl ein.

z.B.:

```
Test  MACRO
   dc.b  \*STRLEN(\1)
   dc.b  "\1"
   ENDM

   Test  <Example>
```
wird zu:

```
   dc.b  7
   dc.b  "Example"
```

3. *LEFT(<String>,n)

Diese Funktion trägt die ersten n Zeichen des Strings in die Macrozeile ein.

```
Test  MACRO
   dc.b  "\*LEFT("\1",4)"
   ENDM

Test  <Example>
```
wird zu:

```
dc.b  "Exam"      ; die ersten 4 Zeichen
```

4. *RIGHT(<String>,n)

Diese Funktion trägt die letzten n Zeichen des Strings in die Macrozeile ein.

5. *MID(<String>,s,n)

Diese Funktion übernimmt ab Position s aus dem String n Zeichen und trägt diese in die Macrozeile ein.

z.B.:

```
Test  MACRO
   dc.b  "\*MID("\1",4,2)"
   ENDM

Test  <Example>
```
wird zu :

```
dc.b  "mp"  ; ab dem 4.Zeichen 2 Zeichen übernehmen
```

6. *UPPER(<String>)

Diese Funktion wandelt alle Buchstaben des Strings in Großbuchstaben um.

```
Test  MACRO
   dc.b  "\*UPPER("\1")"
   ENDM

Test  <Example>
```

wird zu:

```
dc.b  "EXAMPLE"
```

3.4 Bedingte Assemblierung

Der Assembler ermöglicht Ihnen, bestimmte Programmteile in Abhängigkeit einer Bedingung zu assemblieren. Anwendungsgebiete dafür sind z.B. das Assemblieren von Include-Dateien in nur einem Pass oder die wiederholte Assemblierung mit rekursiven Macros. Der Aufbau einer IF-Konstruktion gestaltet sich wie folgt:

```
IFcc    <Term1>[,<Term2>]
...             ;Befehle, die assembliert werden,
                ;wenn das Ergebnis WAHR war
ELSE            ; optional !
...             ;Befehle, die assembliert werden,
                ;wenn das Ergebnis FALSCH war
ENDIF
```

Für cc sind alle Conditioncodes erlaubt, die Sie von den bedingten Sprungbefehlen des MC 68000 her kennen.

```
    IFEQ    1,2     ;ist immer falsch !
    IFEQ    1-1     ;ist immer wahr !
    IFCS    1,2     ;ist falsch, da 2 nicht größer als 1 ist

CALL   MACRO
    IFEQ    NARG-2  ;wurden genau 2 Parameter übergeben?
    move.l \2,a6    ;ja,dann diesen Befehl assemblieren!
    ELSE
    move.l 4.w,a6   ;sonst diesen
    ENDIF
    jsr \1(a6)
    ENDM

CALL    -30,DosBase
```

führt also zu:

```
   move.l  DosBase,a6
   jsr -30(a6)
```

während

```
CALL    -30
```

zu

```
    move.l  4.w,a6
    jsr -30(a6)
```

wird.

Auch wiederholte Assemblierung in Verbindung mit rekursiven Macros ist sehr leicht möglich:

```
TABLE  MACRO
    dc.b    CNT
```

```
CNT=CNT+1
        IFCC    CNT,100
    TABLE       ;ruft sich solange auf, bis CNT > 100!

    ENDIF
    ENDM

CNT=1
    TABLE       ;erzeugt eine Tabelle aus Bytes von
                ;1 bis 100
CNT=50
    TABLE       ;erzeugt eine Tabelle aus Bytes von
                ;50 bis 100
```

Weiterhin gibt es noch einige zusätzliche Direktiven zum bedingten Assemblieren:

```
IFC <String1>,<String2>
```

vergleicht zwei Strings und ist TRUE, wenn die Strings gleich sind. Es wird nicht zwischen Groß- und Kleinschreibung unterschieden, beide Strings müssen gleich lang sein.

```
IFNC <String1>,<String2>
```

ist FALSE, wenn Strings gleich sind und die gleiche Länge haben.

```
IFD  <Variable>
```

ist TRUE, wenn die Variable definiert ist.

```
IFND <Variable>
```

ist FALSE, wenn die Variable bereits definiert ist.

3.5 ALIGNment

Insbesondere bei Opcodes ist es zwingend notwendig, daß sie an einer geraden Adresse beginnen. Auch Datenbereiche, die Wort- oder Langwortdaten enthalten, sollten an geraden Adressen beginnen. Zudem verlangen viele DOS-Funktionen, daß Strukturen an durch 4 teilbaren Adressen beginnen.

Um dies sicherzustellen, existieren folgende Direktiven:

```
CNOP    <D>,<A>
```

`CNOP` bewirkt, daß die Adresse durch `<A>` teilbar gemacht wird. Danach wird noch `<D>` addiert, so daß Sie auch erreichen können, daß z.B. zwei Bytes hinter einem ALIGNment weiterassembliert wird. Eventuelle Füllbytes werden mit `NULL` gefüllt.

```
CNOP 0,2        ; der nächste Befehl steht an einer geraden ; Adresse
CNOP 0,4        ; der nächste Befehl steht an einer durch
                ; 4 teilbaren Adresse
```

```
CNOP 2,4          ; der nächste Befehl steht 2 Bytes hinter
                  ; einer durch 4 teilbaren Adresse
```

Höhere ALIGNments (größer 8) sind nicht sinnvoll, da die Startadresse eines Programmes im Speicher maximal durch 8 teilbar sein kann.

```
ALIGN.W       ; hat die gleiche Wirkung wie CNOP 0,2
ALIGN.L       ; wie CNOP 0,4
EVEN          ; wie ALIGN.W (gerade Adresse)
```

Ab dem MC 68020 dürfen Wort- und Langwortdaten auch an ungeraden Adressen stehen (aber keine Opcodes !). Sie können deshalb Fehlermeldungen bei Worten bzw. Langworten an ungeraden Adressen unterbinden:

```
ODDOK     ; erlaubt Worte und Langworte an ungeraden Adressen
ODDERR    ; gibt die Fehlermeldung bei ungeraden Adressen
          ; wieder frei (voreingestellt)
```

Opcodes an ungeraden Adressen führen aber auch bei **ODDOK** zu einer Fehlermeldung!

3.6 Sektionierung

Programme können in Sektionen, sogenannte Hunks zerlegt werden. Beim Laden des Programmes werden diese Sektionen dann im Speicher verteilt und es muß somit kein großer Speicherblock für das Gesamtprogramm frei sein. Außerdem gibt es sogenannte BSS-Sektionen, die uninitialisierte Daten (NULL) enthalten und nicht mit in der Objektdatei abgespeichert werden. Ihr Programm darf maximal 256 Sektionen enthalten. Der Standardbefehl zur Einleitung einer neuen Sektion heißt:

```
SECTION <Name>,<Typ>[,<Memory>]SECTION <Name>,<Typ>[_<Mem>]
```

Der **<Name>** wird nur berücksichtigt, wenn Sie ein Link-File erzeugen möchten. Es gibt drei Sektionstypen:

CODE	für den Programmcode oder für Daten
DATA	für vorinitialisierte Daten (z.B. Texte)
BSS	für Daten, die erst während des Programmlaufs anfallen
TEXT	hat die gleiche Wirkung wie CODE.
Optional	ist die Angabe des Speichertyps. Erlaubt sind:
CHIP bzw. _C	CHIP-RAM
FAST bzw. **_F**	FAST-RAM
PUBLIC bzw. **_P**	jeder Speicher wird akzeptiert (voreingestellt)

```
SECTION "",DATA,CHIP
```
bzw.
```
SECTION "",DATA_C
```

erzeugt eine DATA-Sektion, die in das CHIP-RAM gelegt wird.

```
SECTION "",CODE,FAST
```

bzw.

```
SECTION "",CODE_F
```

erzeugt eine CODE-Sektion, die nach dem Laden im FAST-RAM liegen wird. Wohin der Assembler diese Sektion legt, ist vom zur Verfügung stehenden Speicher abhängig, FAST-RAM läßt sich also beim Assemblieren nicht erzwingen!

Um sich Tipparbeit zu sparen, können Sie auch folgende Schlüsselworte anstelle des SECTION-Befehles verwenden:

```
CODE    ; = SECTION "",CODE
TEXT    ; = SECTION "",CODE
DATA    ; = SECTION "",DATA
BSS     ; = SECTION "",BSS
```

Die Schlüsselworte **CODE**,**TEXT**,**DATA** und **BSS**, die Sie zur Einleitung einer neuen Sektion alternativ zum SECTION-Befehl verwenden konnten, existieren nicht mehr.

Gleichnamige Sektionen werden zusammengefaßt. Gleichnamige Sektionen unterschiedlichen Typs oder mit unterschiedlichen Speicherattributen erzeugen eine Fehlermeldung!

Achtung: Im Objektfile erhalten die Sektionen aus Ihrem Programm unter Umständen eine andere Reihenfolge. Bei der Erzeugung des Objektfiles werden die Programmsektionen nach dem Typ geordnet. Erst werden alle **CODE**-, dann die **DATA**- und zum Schluß die **BSS**-Sektionen ins Objektfile geschrieben. Leere Sektionen, auf die nicht zugegriffen wird, werden nicht mit ins Objektfile geschrieben!

Direktiven zur Steuerung der Code-Erzeugung

MC68000

Dieser Zustand ist voreingestellt, es wird Code erzeugt, der auf allen Prozessoren lauffähig ist. Alle Befehle des 68010,68020... sind verboten.

MC68010

Die zusätzlichen Befehle des MC 68010 werden erlaubt.

MC68020

Die zusätzlichen Befehle und Adressierungsarten des MC 68020 werden erlaubt.

MC68030

Wirkt wie MC68020, es werden aber auch die Befehle zur Programmierung der im 68030 integrierten MMU zugelassen.

MC68881 / MC68882

Mit diesen Direktiven werden die Befehle der FPU zugelassen. Möchten Sie die erweiterten Adressierungsarten (s.o.) nutzen, müssen Sie diese noch mit MC68020(MC68030) freigeben.

MC68851

Diese Direktive erlaubt die Programmierung der MC68851-MMU. Achten Sie bitte darauf, daß alle MMU-Befehle priviliegiert sind!

Die Direktive MC68000 macht alle obigen Freigaben wieder rückgängig. Benutzen Sie die Direktiven vorsichtig, es kann nämlich sein, daß auch wenn Sie keine Befehle oder Adressierungsarten z.B. des MC68020 nutzen, Code für den MC68020 entsteht:

```
   MC68000
   move.l   (Test,PC),d0

   MC 68020
   move.l   (Test,PC),d0
   rts

Test  dc.l  0
```

Die beiden gleichen Befehle ergeben unterschiedlichen Code! Beide enthalten eine Vorwärtsreferenz, im zweiten wird ein 32-Bit-Displacement erzeugt!

3.7 Erzeugung von absolutem Code

Mit der Direktive ORG können Sie die Adresse angeben, ab der ein Programm lauffähig sein soll. Nach dem Auftreten von ORG werden für absolute Adressierungen keine Reloc-Informationen angelegt! Vor ORG dürfen keine codeproduzierenden Direktiven stehen, ORG sollte direkt auf einen Sektionsbefehl folgen.

Alternativen zu ORG:

```
ORIGIN
RORG
RORIGIN
```

z.B.:

```
   Section  "Test",Code

   ORG   $40000

L1 lea   L1,a0    ; eine absolute Adressierung, kein Reloc!
   rts
```

`L1` hat den Wert `$40000`, nicht `NULL` wie bei relozierbaren Programmem !

3.8 Konstanten

Um Daten in Ihrem Programm abzulegen, verwenden Sie die Direktive

```
DC.x  <Konstante>[,<Konstante>,...]
```

`x` gibt dabei die Länge an und kann B für Byte-Konstanten, `W` für Wort-Konstanten bzw. `L` für Langwort-Konstanten sein. Mehrere Konstanten werden dabei durch Kommata getrennt. Die Konstante darf ein mathematischer Ausdruck, eine Variable oder ein ASCII-Text, der in Hochkommata bzw. Anführungszeichen eingeschlossen sein muß, sein.

```
dc.b  0,1,2,3,4,5,6,7,8
```
wird assembliert zu:

```
00 01 02 03 04 05 06 07 08
```

```
dc.w  0,1,2,3,4
```
wird assembliert zu:

```
00 01 00 02 00 03 00 04
```

```
dc.b  "dos.library",0
```
wird assembliert zu:

```
64 6F 73 2E 6C 69 62 72 61 72 79 00
```

Für die Ablage von Texten gibt es noch zwei spezielle Direktiven, die einige Schreibarbeit sparen können:

CSTRING <String>[,<String>,...]

Legt den/die String(s) byteweise im Speicher ab und fügt ein NULL-Byte an.

```
CSTRING dos.library
```

hat die gleiche Funktion wie:

```
dc.b "dos.library",0
```

PSTRING <String>[,<String>,...]

Legt den/die String(s) byteweise im Speicher ab. Das erste Byte ist hierbei jedoch nicht das erste Zeichen, sondern enthält die Länge des folgenden Strings:

```
PSTRING DF0
```
wird assembliert zu:

```
03 44 46 30 3A
   ^ ^ Länge des folgenden Strings (3 Byte)
```

Für die DC.x-Direktive existieren einige Alternativen:

```
DB für DC.B
DW für DC.W
DL für DC.L
```

Die Direktive DC erlaubt nun auch die Angabe der Operandenlänge der FPU-Befehle.
Vorher sollten Sie jedoch die Erzeugung von FPU-Codes aktivieren. Hier eine Aufstellung aller Möglichkeiten:

`DC.B`	Bytes
`DC.W`	Words
`DC.L`	Longwords
`DC.S`	IEEE-Single, ein Langwort
`DC.D`	IEEE-Double, zwei Langworte
`DC.X`	IEEE-Extended, drei Langworte
`DC.P`	decimal packed, drei Langworte (nur mit FPU möglich !)

Format der Fließkomma-Ausdrücke:

```
[+|-]xxx.yyy[E[+|-]zzz]
```

z.B.:

```
     3.14e3
-50.10e20
50e-11
     3.1415927*2.5
```

Beispiel :

```
DC.X  3.1415927
```

legt 3.1415927 im IEEE-Extended-Format (3 Langworte) in den Speicher

```
DC.S  3.1415927
```

legt 3.1415927 im IEEE-Single-Format (3 Langworte) in den Speicher

Kommen in Ihrem Programm Fließkomma-Konstanten vor, müssen auf der Boot-Disk im LIBS-Verzeichnis die mathieeedoubbas.library und die mathieeedoubtrans.library vorhanden sein. Diese Libraries werden zur Erzeugung der verschiedenen FPU-Zahlenformate verwendet und arbeiten automatisch mit einer eventuell vorhandenen FPU.

3.9 Reservierte Speicherblöcke

DS.x <Größe>[,<Füllwert>]

Mit dieser Direktive wird ein Block definiert, der mit `<Füllwert>` initialisiert werden kann. Für `x` sind wieder `B` (Bytes), `W` (Words) und `L` (Langworte) erlaubt.

```
DS.B  10,-1
```

definiert einen Block aus 10 Bytes, die mit $FF initialisiert werden.

```
DS.L  3
```

hat die gleiche Funktion wie:

```
DC.L  0,0,0
```

Der Sonderfall `DS.W 0` bringt die Adresse wieder auf Wortgrenze, hat damit also die gleiche Bedeutung wie `CNOP 0,2`. Analog dazu macht `DS.L 0` die Adresse durch 4 teilbar (`CNOP 0,4`). An Stelle von `DS` können Sie auch `BLK` oder `DCB` schreiben.

Auch bei der Direktive `DS` (und ihren Alternativen) ist die Angabe von FPU-Operandenlängen möglich.

```
DS.X  40
```

reserviert Platz für 40 Extended-Zahlen (40*3=120 Langworte)

3.10 Definition einer Variablen

Variablen sind wie unter 3.2.2. beschrieben grundsätzlich redefinierbar. Mit der Direktive EQUwird einer Variablen ein Wert zugewiesen. Der Name der Variablen muß in der ersten Spalte beginnen. Der Wert der Variable muß ein mathematischer Ausdruck sein.

```
Exec    EQU 4
```

weist der Variablen `"Exec"` den Wert 4 zu.

```
CNT     EQU CNT+1
```

erhöht den Wert der Variablen `"CNT"` um 1.

Anstelle von `EQU` können Sie auch `"="`, `"=="` oder `"SET"` schreiben:

```
Exec    =   4
Exec    ==  4
Exec    SET 4
```

3.11 Definition von Strukturoffsets

Zum Erstellen eigener Strukturen oder zur Definition der Strukturen des Betriebssystems stellt Ihnen der Assembler einen internen Zähler zur Verfügung. Ein Feld einer Struktur wird wie folgt definiert:

```
<Name> RS.x     <Anzahl>
```

x kann B für Bytes, W für Words und L für Langworte annehmen. Die Anzahl bestimmt, aus wievielen Bytes (Words oder Langworten) das Feld besteht. Die RS-Direktive weist der Variablen den aktuellen RS-Zähler zu und erhöht diesen in Abhängigkeit der Größe x und der Anzahl.

```
RSSET <Wert>
```

Setzt den internen Zähler auf einen bestimmten Anfangswert.

```
RSRESET
```

Setzt den internen Zähler auf 0. Der aktuelle Wert dieses Zählers läßt sich der Variable `__RS` entnehmen (siehe 3.2.2.5.2).

```
RSRESET                 ;löscht den Zähler
XYZ     RS.L    1       ;weist der Variablen XYZ den Wert 0 zu
                        ;und erhöht __RS um 4 (ein Langwort)
```

Definieren Sie Worte oder Langworte, wird geprüft, ob der Zähler ungerade ist. Wenn ja, wird er vorher um eins erhöht, so daß Words und Langworte immer gerade Offsets erhalten !

```
RSRESET                 ; __RS = 0
XY      RS.B    1       ; __RS = 1
XZ      RS.L    1       ; XZ enthält den Wert 2 !
                        ; __RS = 6
```

3.12 Externe Quelltexte

Der Assembler unterstützt die Möglichkeit, Quelltexte von externen Speichermedien in das Hauptprogramm einzubinden.

```
INCLUDE <Datei>
```

Leitet an dieser Stelle die Assemblierung in die angegebene Datei um. Include-Dateien dürfen beliebig tief geschachtelt werden und beliebig lang sein. Es existieren jedoch einige Konventionen bezüglich des Gebrauchs von Include-Dateien:

- IF-Konstruktionen, die in der Datei geöffnet werden, müssen in der gleichen Datei auch wieder geschlossen werden.
- Macrodefinitionen müssen in der gleichen Datei beendet sein, in der sie begonnen wurden.
- Include-Dateien, die Programm (Code) enthalten, müssen in allen Passes INCLUDEt werden.

Beispiele:

```
INCLUDE exec/types.i
INCLUDE exec/node.i
```

Der Assembler sucht normalerweise im Hauptverzeichnis der Boot-Disk nach den Include-Dateien. Um dem Assembler weitere Verzeichnisse bekannt zu machen, verwenden Sie bitte den Befehl:

```
INCDIR <Pfad>
```

Fügt den angegebenen Pfad in die Suchliste der Include-Dateien ein. Bis zu maximal 32 Pfade können mit diesem Befehl angemeldet werden.

```
INCDIR  DF0:
INCDIR  DF1:
INCDIR  DF1:Includes
```

bewirkt, daß der Assembler in diesen drei Verzeichnissen nach einer zu INCLUDEnden Datei sucht.

3.13 Externe Rohdateien

Möchten Sie z.B. Graphik- oder Sounddaten in Ihre Programme einbinden, und sind diese zu umfangreich, um sie in `DC.x`-Zeilen zu konvertieren, können Sie die Daten direkt von Disk in den Programmcode einbinden.

```
INCBIN <Datei>[,<Länge>]
```

Die angegebene Datei wird direkt in den Programmcode integriert. Die Längenangabe ist optional und gibt an, wieviele Bytes der Datei geladen werden sollen. Geben Sie die Länge nicht an, oder ist sie größer als die Dateilänge, so wird die Datei komplett geladen. Nach dem Laden wird die Adresse nicht wieder auf Wortgrenze gebracht, d.h. Sie müssen selbst darauf achten, daß folgende Codes an geraden Adressen beginnen.

3.14 Copper-Assembler

Der Assembler unterstützt die direkte Programmierung von Copperbefehlslisten. Der Copper ist einer der Coprozessoren des Amiga und steuert wichtige Parameter der Bildausgabe in Abhängigkeit der aktuellen Rasterstrahlposition. Der Befehlssatz des Coppers ist zwar nicht besonders umfangreich, er umfaßt nur drei Befehle, die aber ausreichen, sogar Schleifen oder komplexe sich selbst modifizierende Befehlslisten zu programmieren.

Die Befehle im einzelnen sind:

`CMOVE` ; Beschreiben eines Registers

`CWAIT` ; auf eine bestimmte Position warten

`CSKIP` ; testen, ob die angegebene Position schon erreicht war und wenn ja, den nächsten Befehl übergehen

In einer Copperliste hat jeder Befehl eine feste Länge von 4 Bytes. Das Bit 0 beider Befehlsworte dient zum eindeutigen Erkennen der einzelnen Befehle:

erstes Wort	zweites Wort	
0	x	CMOVE (x = 0|1)
1	0	CWAIT
1	1	CSKIP

Aufbau der Befehle im einzelnen:

Bit	CMOVE		CWAIT		CSKIP	
	1.Wort	2.Wort	1.Wort	2.Wort	1.Wort	2.Wort
15	RB15	DT15	VP7	BFD	VP7	BFD
14	RB14	DT14	VP6	VM6	VP6	VM6
13	RB13	DT13	VP5	VM5	VP5	VM5
12	RB12	DT12	VP4	VM4	VP4	VM4
11	RB11	DT11	VP3	VM3	VP3	VM3
10	RB10	DT10	VP2	VM2	VP2	VM2
9	RB9	DT9	VP1	VM1	VP1	VM1
8	RB8	DT8	VP0	VM0	VP0	VM0
7	RB7	DT7	HP8	HM8	HP8	HM8
6	RB6	DT6	HP7	HM7	HP7	HM7
5	RB5	DT5	HP6	HM6	HP6	HM6
4	RB4	DT4	HP5	HM5	HP5	HM5
3	RB3	DT3	HP4	HM4	HP4	HM4
2	RB2	DT2	HP3	HM3	HP3	HM3
1	RB1	DT1	HP2	HM2	HP2	HM2
0	0	DT0	1	0	1	1

`RB15...RB0:` Offset des Zielregisters (z.B.: $0096 für DMACON)
`DT15...DT0:` Datenwort, das in das Register übertragen wird
`VP7 ...VP0:` vertikale Position des Rasterstrahls
`HP8 ...HP1:` horizontale Rasterstrahlposition
`VM7 ...VM0:` Maske für vertikale Rasterstrahlposition
`HM8 ...HM1:` Maske für horizontale Rasterstrahlposition
`BFD :` Blitter-Finish-Disable Bit
(0 = auf Ende der Blitteroperation warten)

Wie Sie sehen, ist der Aufbau der Befehle `CWAIT` und `CSKIP` bis auf das Bit 0 des zweiten Wortes identisch. Doch nun zur Syntax der einzelnen Befehle:

Es ist zwingend notwendig, daß Copper-Befehlslisten immer im CHIP-RAM stehen. Weiterhin müssen Copper-Befehle an geraden Adressen stehen!

```
CMOVE  #<Data>,<Register>  oder  CMOVE #<Data>,(<Register>)
```

<Data> ist das Datenwort, welches in das entsprechende Register übertragen werden soll.

```
CMOVE #$0000,$180       ; setzt die Hintergrundfarbe auf schwarz
CMOVE #$7FFF,$096       ; schaltet alle DMA's aus ($096 = DMACON)
                        ; Vorsicht! Danach funktioniert im Prinzip
                        ; nichts mehr! (kein Bild, kein Ton, kein
                        ; Diskzugriff ...)
```

```
CLMOVE #<Data>,<Register>  oder  CLMOVE #<Data>,(Register)
```

<Data> ist hierbei ein Langwort, das auf zwei Register verteilt wird. **CLMOVE** erzeugt also zwei einzelne **CMOVE**-Befehle. Das obere Wort von <Data> wird in das angegebene Register übertragen und das untere Wort in <Register>+2.

```
CLMOVE #0,$180          ;setzt die Hintergrundfarbe und Farbe
                        ;#1 auf schwarz (0)
```

```
CWAIT  <VPos>,<HPos>[{<VMask>,<HMask>}BFD]
```

VPos: vertikale Position von $00 bis $FF
HPos: horizontale Position von $02 bis $1FE
VMask: Maske für Vertikalposition
HMask: Maske für Horizontalposition
BFD : BFD-Bit 0 | 1

```
CWAIT $2c,0{}0
```

Wartet auf die Position **$2C** (Vert.) 0 (Horiz.) und auf das Ende der letzten Blitteroperation (**BFD=0**). Als Maske wird voreingestellt **$7F** (Vert.) und **$1FE** (Horiz.) eingesetzt.

```
CWAIT $2c,0
```

Wartet auf Postition **$2C**,0 jedoch nicht auf das Ende der letzten Blitteroperation.

Die Syntax des **CSKIP**-Befehles ist identisch mit der des **CWAIT**-Befehles. Haben Sie bemerkt, daß es keinen **END**-Befehl zum Abschluß einer Copper-Liste gibt? Um eine Copperliste zu beenden, verwendet man einen **CWAIT**-Befehl, der auf eine unmögliche Rasterposition wartet.

Z.B.:

```
CWAIT  $FF,$1FE
```

Die horizontale Position **$1FE** existiert nicht, und der Copper wartet sozusagen umsonst. Ist das Bild fertig dargestellt, wird jedoch die Copperliste neu gestartet, so daß der Copper sich nicht „aufhängen" kann. Um die Copperliste zu beenden, können Sie aber auch die folgende Direktive verwenden:

```
CEND
```

CEND erzeugt nichts anderes, als den oben beschriebenen unmöglichen **CWAIT**-Befehl.

Ein Problem ergibt sich allerdings noch aus dem Aufbau der **CWAIT**- und **CSKIP**-Befehle. Für die Angabe der vertikalen Position stehen nur 8 Bits (**VP0...VP7**) zur Verfügung. Ein PAL-Bild hat aber mehr als 256 Zeilen. Die übrigen Zeilen erreichen Sie, wenn Sie auf das Ende der Zeile 255 warten. Die übrigen Zeilen sind dann wieder ab Zeile 0 ... erreichbar.

Copperbefehle müssen mit der Direktive **COPPER** erst zugelassen werden. Die Erzeugung von Copper-Befehlen kann mit **ENDCOP** wieder verboten werden.

z.B.:

```
Color00=$180
Color01=$182
Color02=$184
Color03=$186

   COPPER

   CMOVE #$AAA,Color00
   CMOVE #$000,Color01
   CMOVE #$FFF,Color02
   CMOVE #$68B,Color03
   CEND

   ENDCOP
```

3.15 Externe Symbole

Um dem Assembler bzw. dem Linker externe Symbole bekannt zu machen, existieren folgende Direktiven:

```
XREF <Symbol1>[,<Symbol2>...]
```

Diese Direktive teilt dem Assembler mit, daß die angegebenen Symbole nicht in diesem Codemodul definiert werden, sondern aus einem anderen Modul importiert werden. Der Linker setzt dann an die Stellen, an denen dieses Symbol benutzt wird, die richtigen Werte ein. Diese Direktive dürfen Sie also nur benutzen, wenn Sie linkfähigen Code erzeugen möchten. Es werden 32-Bit, 16-Bit und 8-Bit Referenzen auf ein externes Symbol unterstützt.

Beispiel für eine 32-Bit-Referenz:

```
XREF    Main
Start   jsr Main          ;32-Bit-Referenz
...
```

Beispiel für eine 16-Bit-Referenz:

```
XREF    _LVOOpenLibrary
...
    jsr _LVOOpenLibrary(a6) ;16-Bit-Referenz
...
```

Beispiel für eine 8-Bit-Referenz:

```
XREF    LN_SIZE
...
moveq   #LN_SIZE,d0       ;8-Bit-Referenz
...
```

XREF führt zu einer Fehlermeldung, wenn Sie ausführbaren Code erzeugen möchten.

```
XDEF    <Symbol1>[,<Symbol2>...]
```

Exportiert die angegebenen Symbole, so daß sie von anderen Codemodulen benutzt werden können. Dabei wird zwischen relocatiblen Labeln und Variablen unterschieden.

```
DEF  Var     ;diese Variable wird anderen Modulen
             ;bekannt gemacht
XDEF _Main   ;_Main wird anderen Modulen bekannt
             ;gemacht
Var EQU 267
_Main   moveq   #0,d0
...
...
rts
```

Zusätzlich zu den oben benutzten Direktiven können Sie auch folgende Alternativen benutzen:

1. für **XREF:** EXTERN
2. für **XDEF:** PUBLIC
   ENTRY
   GLOBAL
   GLOBL

3.16 Codearten

Der Assembler unterscheidet zwischen drei verschiedenen Code-Arten:

1. Lauffähiger Code

Diesen Code werden Sie im Zusammenhang mit dem C-Compiler eher selten erzeugen. Immer, wenn Sie die Entscheidung über die Art des erzeugten Codes dem Assembler überlassen und die Direktive XREF bzw. deren Alternativen nicht benutzen, wird lauffähiger Code erzeugt.

2. Linkbarer Code (Option l)

Verwenden Sie in Ihrem Quelltext externe Referenzen (XREF), wird beim Abspeichern ein Link-File erzeugt, das Sie mit anderen Modulen zu einem lauffähigen Programm linken können. Linkfiles sind weder vom CLI noch von der Workbench startbar. Große Programmprojekte werden sinnvollerweise in Module zerlegt, die erst durch den Linker zu einem fertigen Programm zusammengefügt werden.

3. Absoluter Code (Option a)

Haben Sie ein lauffähiges Programm erzeugt, so können Sie hiermit absoluten Code abspeichern. Sie werden zu der Eingabe der Adresse aufgefordert, ab der das Programm später lauffähig sein soll. Absoluten Code können Sie allerdings weder vom CLI noch von der Workbench starten. Diese Option ist nur sinnvoll, wenn Sie von vornherein wissen, an welcher Adresse das Programm später laufen soll.

3.17 Assemblerbetriebsarten

Vor dem Assemblieren eines Quelltextes können Sie zwischen drei verschiedenen Betriebsarten wählen:

1. Normal

Der Assembler führt zwei Passes durch. Der Code wird nicht optimiert. Es entsteht Code, den Sie abspeichern können.

2. Optimieren (Option o)

Hiermit schalten Sie den Optimierungsmodus ein. Alle aktivierten Optimierungen werden versucht, es werden soviele Passes wie nötig durchgeführt, d.h. der Assembler erzeugt erst Code, wenn alle Optimierungsmöglichkeiten ausgeschöpft sind. Mindestens 3 Passes werden durchgeführt, nach oben sind im Prinzip keine Grenzen gesetzt. Die Anzahl der durchgeführten Passes hängt vom Programm ab. Es kann also durchaus vorkommen, daß der Assembler 16 Passes assembliert. Sie erhalten allerdings ein in Länge und Laufzeit hochoptimiertes Programm, so daß sich die Wartezeit lohnt.

3.18 Mögliche Optimierungen

Haben Sie die Multi-Pass-Optimierung aktiviert,wird folgendes optimiert:

```
MOVE.L » MOVEQ
```

Alle `MOVE.L`-Befehle, deren Langwort-Konstante im Bereich `-$80<=K<=$7F` liegt und in ein Datenregister übernehmen, werden in einen `MOVEQ`-Befehl umgewandelt. Sie sparen dabei 4 Bytes und 8 Taktzyklen (ein `MOVE.L #K,Dn` benötig 12 Taktzyklen gegenüber 4 Taktzylken für den `MOVEQ`-Befehl).

```
20 3C 00 00 00 3C    MOVE.L #$3C,D0
```

wird optimiert zu:

```
70 3C                MOVEQ  #$3C,d0
```

```
CLR.L Dn » MOVEQ #0,Dn
```

Alle `CLR.L Dn`-Befehle werden in ein `MOVEQ #0,Dn` umgewandelt. Der Opcode wird dabei nicht kürzer, ein `MOVEQ` ist aber um 2 Taktzyklen schneller.

```
LABEL » LABEL(PC)
```

Adressieren Sie ein Label, das in der gleichen Sektion liegt und nicht weiter als 32768 Bytes vom Befehl entfernt ist absolut, wird die Adressierung wenn möglich nach PC-Relativ umgewandelt. Der Befehl wird um 2 Bytes kürzer, die Ausführungszeit verkürzt sich um 4 Taktzyklen und die Programmdatei auf Disk wird um 4 Bytes kürzer (Relocoffset).

```
MOVE.L DosBase,a6
```

wird zu

```
MOVE.L DosBase(PC),a6
```

aber:

```
MOVE.L d0,DosBase
```

kann nicht optimiert werden, da PC-relative Adressierung nur im Quelloperand möglich ist.

```
XXXX.L » XXXX.W
```

Absolut lange Adressierungen werden in absolut kurz gewandelt. Das ist natürlich nur möglich, wenn die Adresse zwischen `-$8000` und `$7FFF` liegt. Ein Beispiel für diese Optimierung ist das Holen der Exec-Basis:

```
MOVE.L 4,A6
```

wird zu:

```
MOVE.L 4.W,A6
```

Der Befehl wird um zwei Bytes kürzer und um 4 Taktzyklen schneller.

```
Bcc.L » Bcc.S
```

Alle Branch-Befehle, deren Ziel im Bereich von -$80 und $7F liegt, werden in einen Short-Branch gewandelt. Der Befehl wird um zwei Bytes kürzer und um 4 Taktzyklen schneller, wenn die Verzweigung nicht ausgeführt wurde (außer **BRA** und **BSR**).

```
00(An) » (An)
```

Adressierungen mit der Distanz 0 werden in die Adressierung **ARI** ohne Distanz umgewandelt. Sie sparen dadurch 2 Bytes und die Ausführung wird um 4 Taktzyklen beschleunigt.

```
ADD/SUB » ADDQ/SUBQ
```

Addieren Sie eine Konstante von 1 bis einschl. 8, wird der **ADD**-Befehl (**SUB**) in einen **ADDQ**-Befehl (**SUBQ**) optimiert. Sie sparen dabei 2 Bytes und der Befehl wird um 4 Taktzyklen schneller. Bei Langwortverarbeitung sparen Sie sogar 4 Bytes.

```
CMP #0,<ea> » TST <ea>
```

Befehle, in denen Sie einen Operanden mit 0 vergleichen, werden in einen **TST**-Befehl umgewandelt (außer **CMP #0,An**). Sie sparen 2 Bytes (4 bei Langwortverarbeitung) und 4 Taktzyklen.

```
Label » Label(BASEREG)
```

Absolute Adressierungen werden in relative zu einem Adressregister umgewandelt. Dazu müssen Sie dem Assembler bekanntgeben, welches Adressregister verwendet werden soll und welcher Wert sich in diesem Register befindet. Dazu verwenden Sie bitte den Befehl:

```
BASEREG An,<Wert>
```

z.B.:

```
BASEREG A5,Data
DataDosBase dc.l    0
Handle  dc.l    0
```

führt zu folgenden Optimierungen:

```
move.l  DosBase,a6
```

wird zu

```
MOVE.L (A5),a6
```

```
move.l  Handle,d1
```

wird zu

```
MOVE.L 4(A5),a6
```

Diese Optimierung können Sie mit

```
BASEREG OFF
```

abschalten.

Das ist besonders wichtig, wenn Sie das Register, mit dem optimiert werden soll, initialisieren:

```
BASEREG OFF          ;abschalten
lea Data,a5          ;A5 initialisieren
BASEREG A5,Data      ;ab hier darf optimiert werden
```

3.19 Ausgabedateien

Das Protokoll einer laufenden Assemblierung kann in eine Datei gschrieben werden. Dazu haben Sie drei Möglichkeiten:

1. Listing (-l)

Es wird ein Assemblerlisting erzeugt, das die Quelltextzeile und den entstandenen Code auflistet.

```
ZEILE    PC     CODE                TEXT
    1 00000000 7000                Start moveq #0,d0
    2 00000002 7200                 moveq #0,d1
    3 00000004 41F90000000C         lea Label,a0
    4 0000000A 4E75                 rts
    5 0000000C FFFFFFFF            Label dc.l -1
```

Mit der Direktive NOLIST können Sie die Erzeugung eines Protokolls verbieten und mit LIST wieder erlauben.

Z.B.:

```
NOLIST
include exec/types.i     ;diese Include-Dateien erscheinen
include exec/nodes.i     ;nicht im Listing !
include exec/lists.i
LIST                     ;ab dieser Zeile wird wieder ein
                         ;Listing erzeugt.
```

2. Symboltabelle (-f)

Alle im Quelltext definierten globalen Symbole werden getrennt nach Labeln und Variablen in eine Datei ausgegeben.

3. Fehlerdatei (-u)

Sollten beim Assemblieren Fehler auftreten, werden diese nicht angezeigt, sondern in eine Datei umgeleitet. Die Assemblierung wird dann trotzdem fortgesetzt, so daß Sie alle Fehlermeldungen

erhalten. Es kann jedoch vorkommen, daß einige Fehler Folgefehler sind, da die Assemblierung einer Zeile beim Auftreten eines Fehlers abgebrochen wird.

Anhang A

Die Befehle des MC 68000 / MC 68010

Der folgende Überblick über die Befehle des MC 68000 / MC 68010 umfaßt für jeden Befehl die korrekte Syntax, eine kurze Beschreibung der ablaufenden Operation und der erlaubten Adressierungsarten.

Die verwendeten Abkürzungen haben folgende Bedeutung :

`Dn`	ein Datenregister D0 ... D7
`An`	ein Adressregitsre A0 ... A7
`Rn`	ein beliebiges Adress- oder Datenregister
`X`	das X-Flag
`N`	das N-Flag
`Z`	das Z-Flag
`C`	das C-Flag
`V`	das V-Flag
`<ea>`	effektive Adresse (eine der erlaubten Adressierungsarten)
`.x`	Operandengröße
`#K`	eine Konstante
`cc`	Condition-Codes
`d`	im Befehl angegebene relative Adressdistanz
`RList`	Registerliste z.B. d0-d7/a2-a3/a5

Operandengrößen in eckigen Klammern sind optional, brauchen also nicht angegeben zu werden (z.B. [B]: Der Befehl verarbeitet nur Byte-Operanden) Befehle vor denen ein `"*"` steht, sind nur im Supervisor-Mode erlaubt !

Adressierungsarten :

```
   #K  Konstante unmittelbar
d8(PC,Rn)  PC-Relativ mit Indexregister
  d16(PC)  PC-Relativ
      $.L  Absolut lang
      $.W  Absolut kurz
d8(An,Rn)  ARI mit Distanz und Indexregister
  d16(An)  ARI mit Distanz
    -(An)  ARI mit Predekrement
    (An)+  ARI mit Postinkrement
     (An)  Adressregister indirekt (ARI)
       An  Adressregister direkt
       Dn  Datenregister direkt
```

Befehl		Operation	Flags	000000000011 012345678901
ABCD Dn,Dn	[B]	Ziel=Ziel+Quelle-X BCD	XNZCV	
ABCD -(An),-(An)	[B]	Ziel=Ziel+Quelle-X BCD	XNZCV	
ADD.x Dn,<ea>	BWL	Ziel=Quelle+Ziel	XNZCV	..*******...
ADD.x <ea>,Dn	BWL	Ziel=Quelle+Ziel	XNZCV	**********
ADDA.x <ea>,An	WL	Ziel=Quelle+Ziel		**********
ADDI.x #K,<ea>	BWL	Ziel=Quelle+Ziel	XNZCV	*.*******...
ADDQ.x #K,<ea>	BWL	Ziel=Quelle+Ziel 0<K<9	XNZCV	*********...
ADDX.x Dn,Dn	BWL	Ziel=Quelle+Ziel+X	XNZCV	
ADDX.x -(An),-(An)	BWL	Ziel=Quelle+Ziel+X	XNZCV	
AND.x Dn,<ea>	BWL	Ziel=Quelle^Ziel	XNZCV	..*******...
AND.x <ea>,Dn	BWL	Ziel=Quelle^Ziel	XNZCV	**********
ANDI.x #K,<ea>	BWL	Ziel=Quelle^Ziel	XNZCV	*.*******...
ANDI #K,CCR	[B]	Ziel=Quelle^Ziel	XNZCV	
*ANDI #K,SR	[W]	Ziel=Quelle^Ziel	XNZCV	
ASL.x Dn,Dn	BWL	Ziel um Dn verschoben	XNZCV	
ASL.x #K,Dn	BWL	Ziel um #K verschoben	XNZCV	
ASL <ea>	[W]	Ziel um 1 verschoben	XNZCV	..*******...
ASR.x Dn,Dn	BWL	Ziel um Dn verschoben	XNZCV	
ASR.x #K,Dn	BWL	Ziel um #K verschoben	XNZCV	
ASR <ea>	[W]	Ziel um 1 verschoben	XNZCV	..*******...
Bcc Label		cc=TRUE : PC=PC+d		
BCHG Dn,<ea>		Bit testen und invert.	Z	*.*******...
BCHG #K,<ea>		Bit testen und invert.	Z	*.*******...
BCLR Dn,<ea>		Bit testen und löschen	Z	*.*******...
BCLR #K,<ea>		Bit testen und löschen	Z	*.*******...
BSET Dn,<ea>		Bit testen und setzen	Z	*.*******...
BSET #K,<ea>		Bit testen und setzen	Z	*.*******...
BTST Dn,<ea>		Bit testen	Z	*.*******...
BTST #K,<ea>		Bit testen	Z	*.*******...
BRA Label		PC=PC+d		
BSR Label		PC » Stack,PC=PC+d		
CHK <ea>,Dn	[W]	Dn<0|Dn><ea> » Except.	NZCV	***********
CLR.x <ea>	BWL	<ea>=0	NZCV	*.*******...
CMP.x <ea>,Dn	BWL	Flags=Quelle-Ziel	NZCV	**********

CMPA.x <ea>,An	WL	Flags=Quelle-Ziel	NZCV	**********
CMPI.x #K,<ea>	BWL	Flags=Quelle-Ziel	NZCV	*.******...
CMPM.x (An)+,(An)+	BWL	Flags=Quelle-Ziel	NZCV	
DBcc Dn,Label		cc=FALSE : Dn=Dn-1		
		Dn <> -1 : PC=PC+d		
DIVS <ea>,Dn	[W]	Ziel=Ziel/Quelle	NZCV	*.**********
DIVU <ea>,Dn	[W]	Ziel=Ziel/Quelle	NZCV	*.**********
EOR.x Dn,<ea>	BWL	Ziel=Ziel EOR Quelle	NZCV	*.*******...
EORI.x #K,<ea>	BWL	Ziel=Ziel EOR Quelle	NZCV	*.*******...
EORI.x #K,CCR	[B]	Ziel=Ziel EOR Quelle	XNZCV	
*EORI.x #K,SR	[W]	Ziel=Ziel EOR Quelle	XNZCV	
EXG Rn,Rn		Rn <-> Rn		
EXT.x Dn	WL	verzeichenr. erweitern	NZCV	
ILLEGAL		Illegal Opcode Except.		
JMP <ea>		PC=<ea>		..*..******.
JSR <ea>		PC » Stack,PC=<ea>		..*..******.
LEA <ea>,An		An=<ea>		..*..******.
LINK An,#D		An»Stack,SP»An,SP=SP+D		
LSL.x Dn,Dn	BWL	Ziel um Dn verschoben	XNZCV	
LSL.x #K,Dn	BWL	Ziel um #K verschoben	XNZCV	
LSL <ea>	[W]	Ziel um 1 verschoben	XNZCV	..*******...
LSR.x Dn,Dn	BWL	Ziel um Dn verschoben	XNZCV	
LSR.x #K,Dn	BWL	Ziel um #K verschoben	XNZCV	
LSR <ea>	[W]	Ziel um 1 verschoben	XNZCV	..*******...
MOVE.x <ea>,<ea>	BWL	Quelle » Ziel	NZCV	**********
				*.*******...
MOVE <ea>,CCR	[B]	CCR=<ea>	XNZCV	*.**********
*MOVE <ea>,SR	[W]	SR =<ea>	XNZCV	*.**********
MOVE SR,<ea>	[W]	<ea>=SR		*.*******...
*MOVE USP,An	[L]	An=USP		
*MOVE An,USP	[L]	USP=An		
MOVEA.x <ea>,An	WL	An=<ea>		**********
MOVEM.x RList,<ea>	WL	<ea>=RList		..*.*****...
MOVEM.x <ea>,RList	WL	RList=<ea>		..**.******.
MOVEP.x Dn,D16(An)	WL	Ziel=Dn		
MOVEP.x D16(An),Dn	WL	Dn=Quelle		
MOVEQ #K,Dn	[L]	Ziel=Quelle	NZCV	
MULS <ea>,Dn	[W]	Ziel=Ziel*Quelle	NZCV	*.**********
MULU <ea>,Dn	[W]	Ziel=Ziel*Quelle	NZCV	*.**********
NBCD <ea>	[B]	<ea>=0-<ea>-X	XNZCV	*.*******...
NEG.x <ea>	BWL	<ea>=0-<ea>	XNZCV	*.*******...
NEGX.x <ea>	BWL	<ea>=0-<ea>-X	XNZCV	*.*******...
NOP		Keine Operation		
NOT.x <ea>	BWL	<ea>=0-<ea>-1	NZCV	*.*******...
OR.x Dn,<ea>	BWL	Ziel=Ziel OR Quelle	NZCV	..******...
OR.x <ea>,Dn	BWL	Ziel=Ziel OR Quelle	NZCV	*.**********
ORI.x #K,<ea>	BWL	Ziel=Ziel OR Quelle	NZCV	*.*******...
ORI #K,CCR	[B]	Ziel=Ziel OR Quelle	XNZCV	
*ORI #K,SR	[W]	Ziel=Ziel OR Quelle	XNZCV	
PEA <ea>		<ea> » Stack		..*..******.
*RESET		Resetleitung LOW		
ROL.x Dn,Dn	BWL	Ziel um Dn verschoben	XNZCV	

ROL.x #K,Dn	BWL	Ziel um #K verschoben	XNZCV		
ROL <ea>	[W]	Ziel um 1 verschoben	XNZCV	..*******...	
ROR.x Dn,Dn	BWL	Ziel um Dn verschoben	XNZCV		
ROR.x #K,Dn	BWL	Ziel um #K verschoben	XNZCV		
ROR <ea>	[W]	Ziel um 1 verschoben	XNZCV	..*******...	
ROXL.x Dn,Dn	BWL	Ziel um Dn verschoben	XNZCV		
ROXL.x #K,Dn	BWL	Ziel um #K verschoben	XNZCV		
ROXL <ea>	[W]	Ziel um 1 verschoben	XNZCV	..*******...	
ROXR.x Dn,Dn	BWL	Ziel um Dn verschoben	XNZCV		
ROXR.x #K,Dn	BWL	Ziel um #K verschoben	XNZCV		
ROXR <ea>	[W]	Ziel um 1 verschoben	XNZCV	..*******...	
*RTE		PC und SR vom Stack	XNZCV		
RTR		PC und CCR vom Stack	XNZCV		
RTS		PC vom Stack			
SBCD Dn,Dn	[B]	Ziel=Ziel-Quelle-X BCD	XNZCV		
SBCD -(An),-(An)	[B]	Ziel=Ziel-Quelle-X BCD	XNZCV		
Scc <ea>		Setze <ea>,wenn cc=TRUE		*.*******...	
*STOP #K		SR=#K STOP-Zustand	XNZCV		
SUB.x Dn,<ea>	BWL	Ziel=Ziel-Quelle	XNZCV	..*******...	
SUB.x <ea>,Dn	BWL	Ziel=Ziel-Quelle	XNZCV	**********	
SUBA.x <ea>,An	WL	Ziel=Ziel-Quelle	.....	**********	
SUBI.x #K,<ea>	BWL	Ziel=Ziel-Quelle	XNZCV	*.*******...	
SUBQ.x #K,<ea>	BWL	Ziel=Ziel-Quelle 0<K<9	XNZCV	********...	
SUBX.x Dn,Dn	BWL	Ziel=Ziel-Quelle-X	XNZCV		
SUBX.x -(An),-(An)	BWL	Ziel=Ziel-Quelle-X	XNZCV		
SWAP Dn		Bit 31-16 <-> Bit 15-0	NZCV		
TAS <ea>		<ea> abgefragt » CCR	NZCV	*.*******...	
		Bit 7 von <ea> = 1			
TRAP #Vektor		PC,SR»Stack,PC=(Vektor)			
TRAPV		TRAP wenn V=1			
TST <ea>	BWL	<ea> abgefragt » Flags	NZCV	*.*******...	
UNLK An		SP=An,An vom Stack			

Die zusätzlichen Befehle des MC 68010

Abkürzungen :

CTRLReg Control-Register (DFC,SFC,VBR,USP)

Befehl		Operation	Flags	000000000011
				012345678901
*MOVEC CTRLReg,Rn	[L]	Rn=CTRLReg		
*MOVEC Rn,CTRLReg	[L]	CTRLReg=Rn		
*MOVES Rn,<ea>	BWL	Ziel=Quelle		..*******...
*MOVES <ea>,Rn	BWL	Ziel=Quelle		..*******...
MOVE CCR,<ea>	[B]	<ea>=CCR		*.*******...
RTD #D		PC vom Stack, SP=SP+D		

Der beim MC 68000 nicht privilegierte Befehl **MOVE SR,<ea>** darf ab dem MC 68010 nur noch im Supervisor-Mode benutzt werden !

Anhang B

Die Assemblerdirektiven

BSS
Leitet eine BSS-Sektion ein. Memory-Typ ist MEMF_PUBLIC.

SECTION <Name>,<Typ>,<Memory-Typ>
Leitet eine neue Sektion **<Typ>** ein.

<Name>	:	Name der Sektion
<Typ>	:	Sektionstyp CODE(TEXT), DATA oder BSS
<Memory-Typ>	:	FAST,CHIP oder PUBLIC

NOMEXP
Verbietet die Macroexpansion im Listing.

CEND
Erzeugt einen END-Befehl für eine Copperliste (= **CWAIT -1,-2**).

DATA
Leitet eine DATA-Sektion ein. Memory-Typ ist **MEMF_PUBLIC**.

NOLIST
Unterdrückt die Ausgabe in ein Listing.

ALIGN.x
Bringt die Adresse auf Wortgrenze (**.W**) bzw. auf Langwortgrenze (**.L**).

EQU
Weist einer Variablen einen Wert zu.

XDEF <Variable>[,<Variable>...]
Macht dem Linker die Variable(n) bekannt.

CODE
Leitet eine CODE-Sektion ein. Memory-Typ ist **MEMF_PUBLIC**.

PUBLIC <Variable>[,<Variable>...]
Eine Alternative zu **XDEF**.

DB <Konstante>[,<Konstante>...]
Assembliert Byte-Konstante(n). (= **DC.B**)

IFcc <Term1>[,<Term2>]
Die Assemblierung wird fortgesetzt, wenn **cc=TRUE**. Ist **cc=False**, wird erst nach einem **ELSE** bzw. dem zugehörigen **ENDIF** weiterassembliert.

ENDC
Beendet eine IF-Konstruktion (= **ENDIF**).

DC.x <Konstante>[,<Konstante>...]
Assembliert Konstante(n) der Größe x (**B/W/L**).

PROGRAM <Name>
Gibt der Programmunit den Namen <Name> (nur bei Link-Files).

DCB.x <Größe>,<Füllwert>
Assembliert einen Block aus **<Größe>** Konstanten der Länge x (**B/W/L**) und initialisiert den Block mit **<Füllwert>**.

IDENTIFY <Name>
Eine Alternative zu PROGRAM.

ENDIF
Beendet eine **IF**-Konstruktion.

CLMOVE #<Wert>,<Register>
Erzeugt zwei **CMOVE**-Befehle (Copper). Ein Langwort **<Wert>** wird in **<Register>** und **<Register>+2** übertragen.

ENDM
Schließt eine Macrodefinition ab.

TEXT
Eine Alternative zu **CODE**.

CNOP <Align>,<Dist>
Macht die Adresse durch **<Align>** (2 oder 4) teilbar und addiert **<Dist>**.

CWAIT <YPos>,<XPos>[{YMask,XMask}BFD]
Erzeugt einen **CWAIT**-Befehl (Copper), der auf **YPos [AND YMask]** und **XPos [AND XMask]** und eventuell auf den Blitter (**BFD=0**) wartet.

CMOVE #<Wert>,<Register>
Erzeugt einen **CMOVE**-Befehl (Copper), der den angegebenen Wert in eine Register der Custom-Chips überträgt.

ELSE
Kehrt das Ergebnis einer **IF**-Abfrage um. War das Ergebnis **FALSE**, werden jetzt die auf **ELSE** folgenden Befehle assembliert.

ELSEIF
Eine Alternative zu **ELSE**.

CSKIP <YPos>,<XPos>[{YMask,XMask}BFD]
Erzeugt einen **CSKIP**-Befehl (Copper).

MACRO
Leitet eine Macrodefinition ein.

RSRESET
Setzt den internen Zähler __RS auf NULL.

PSTRING <String>[,<String>]
Der (Die) String(s) werden mit einem führenden Längenbyte assembliert.

PSTR
Eine Alternative zu **PSTRING**.

RSSET <Wert>
Initialisiert den internen Zähler __RS mit **<Wert>**.

INCBIN <Datei>[,<Länge>]
Eine Datei wird in den Programmcode integriert. **<Länge>** gibt an, wieviele Bytes geladen werden sollen.

GLOBAL <Variable>[,<Variable>]
Eine Alternative zu **XDEF**.

INCDIR <Pfad>
Meldet den Pfad als Suchpfad für **INCLUDE**-Dateien an.

DL <Konstante>[,<Konstante>...]
Assembliert Langwort-Konstante(n) (= **DC.L**).

INCLUDE <Datei>
Leitet die Assemblierung in die angegebene Datei um.

SET
Weist einer Variablen einen Wert zu.

ODDERR
Words und Longwords an ungeraden Adressen führen zu einer Fehlermeldung.

BLK.x <Größe>[,<Füllwert>]
Eine Alternative zu **DCB**.

IBYTES <Datei>[,<Länge>]
Eine Alternative zu **INCBIN**.

EQUATE
Eine Alternative zu **EQU**.

EVEN
Macht die Adresse wieder durch zwei teilbar.

XREF <Variable>[,<Variable>]
Die Variable(n) werden importiert (nur in Link-Files).

ENTRY <Variable>[,<Variable>]
Eine Alternative zu **XDEF**.

RS.x <Anzahl>
Addiert zum RS-Zähler **<Anzahl>** Bytes (**B**), Words (**W**) oder Longwords (**L**).

Steht vor **RS.x** eine Variable, wird ihr der alte Wert von **__RS** zugewiesen.

CSTRING <String>[,<String>]
Die Strings werden mit einem abschließenden **NULL**-Byte assembliert.

CSTR <String>[,<String>]
Eine Alternative zu **CSTRING**.

ODDOK
Words und Longwords an ungeraden Adressen sind erlaubt !

OPT
Hat keine Wirkung (aus Kompatibilitätsgründen vorhanden).

IFC <String1>,<String2>
Vergleicht zwei Strings und gibt **TRUE** wenn beide Strings gleich sind und auch die gleiche Länge haben.

END
Beendet die Assemblierung.

IFD <Variable>
Stellt fest, ob die Variable definiert ist und gibt **TRUE** wenn ja.

DS.x <Größe>[,<Füllwert>]
Eine Alternative zu **DCB**.

MEXIT
Beendet vorzeitig die Bearbeitung eines Macros.

EXTERN <Variable>[,<Variable>]
Eine Alternative zu **XREF**.

DL <Konstante>[,<Konstante>...]
Assembliert Wort-Konstante(n) (= **DC.W**).

MEXP
Erlaubt die Macroexpansion im Listing.

LIST
Gibt die Ausgabe in ein Listing wieder frei.

BASEREG

BASEREG OFF verbietet die Optimierung absoluter Adressen relativ zu einem Adressregister. **BASEREG An,<Basis>** gibt die Optimierung relativ zu An mit der **<Basis>** frei.

FAIL <Meldung>
Erzeugt eine Benutzerfehlermeldung

Anhang C: Die Fehlermeldungen

DOS-Fehler beim Dateihandling

032 DOS-Fehler xxx beim LOCK auf Verzeichnis <Name>
033 DOS-Fehler xxx beim Laden der Quelldatei <Name>
034 DOS-Fehler xxx beim Öffnen der Objektdatei <Name>
035 DOS-Fehler xxx beim Schreiben der Objektdatei <Name>
036 DOS-Fehler xxx beim Öffnen der Includedatei <Name>
037 DOS-Fehler xxx beim Öffnen der Binärdatei <Name>
038 DOS-Fehler xxx beim Lesen der Binärdatei <Name>
039 DOS-Fehler xxx beim Schreiben der Prelistingdatei <Name>
040 DOS-Fehler xxx beim Öffnen der Prelistingdatei <Name>
041 DOS-Fehler xxx beim Öffnen Listingdatei <Name>
042 DOS-Fehler xxx beim Lesen der Prelistingdatei <Name>
043 DOS-Fehler xxx beim Schreiben der Listingdatei <Name>
044 DOS-Fehler xxx beim SEEK in der Prelistingdatei <Name>
046 DOS-Fehler xxx beim Öffnen der Fehlerdatei <Name>
047 DOS-Fehler xxx beim Öffnen der Symboltabelle <Name>

Fehler in der Sektionierung

048 Programm enthält zu viele Sektionen (maximal 256)
049 Falscher Sektionstyp (nur CODE,DATA,BSS)
050 Gleichnamige Sektion eines anderen Types existiert bereits
051 Falscher Speichertyp (nur PUBLIC,CHIP,FAST)
052 Gleichnamige Sektion mit anderen Speichertypen existiert bereits
053 Codeproduzierende Direktiven im BSS-Hunk nicht zulässig
054 Vor „ORG" dürfen keine codeproduzierenden Direktiven stehen
055 Basisadresse nach „ORG" muß gerade sein

MACRO-Fehler

056 Falsche Parameterbezeichnung (nur \0...\9 und \A...\Z)
057 Name vor MACRO fehlt
058 Reservierter Name
059 Makroname muß mit einem Buchstaben oder „_" beginnen
060 Makroname kollidiert mit Label
061 Makroname kollidiert mit absolutem Symbol
062 Makroname kollidiert mit importiertem Symbol
063 Makroname kollidiert mit Registersymbol

064 Illegale Redefinition eines Makros
065 Makroname kollidiert mit Registerlistensymbol
066 MACRO darf nicht innerhalb eines Makros vorkommen
067 MACRO ohne zugehöriges ENDM innerhalb der Includedatei
068 MACRO ohne zugehöriges ENDM
069 Vor ENDM darf kein Label stehen
070 Unzulässiges leeres Makro
071 Zu viele Makroparameter (maximal 36)
072 END darf nicht innerhalb eines Makros vorkommen
073 Falsche IF/ENDIF-Paare im Makro
074 MEXIT darf nur innerhalb eines Makros vorkommen
075 Nach Schlüsselwort muß unmittelbar „(„ folgen
076 Makrofunktion nicht implementiert
077 Makrofonktionen zu tief geschachtelt (maximal 16fach)
078 Argument einer Makrofunktion muß mit „)“ enden
079 VALOF() darf nur innerhalb eines Makros vorkommen
080 STRLEN() darf nur innerhalb eines Makros vorkommen
081 LEFT() darf nur innerhalb eines Makros vorkommen
082 RIGHT() darf nur innerhalb eines Makros vorkommen
083 MID() darf nur innerhalb eines Makros vorkommen
084 UPPER() darf nur innerhalb eines Makros vorkommen
085 *VALOF() : Ausdruck enthält undefinierte Symbole
086 *VALOF() : Argument muß ein Symbol sein
087 *STRLEN() : Fehlende Argumente
088 *LEFT() : Erster Operand fehlt
089 *LEFT() : Zweiter Operand fehlt
090 *LEFT() : Zweiter Operand enthält undefinierte Symbole
091 *LEFT() : Zweiter Operand muß ein absoluter Ausdruck sein
092 *LEFT() : Zweiter Operand muß größer als NULL sein
093 *RIGHT() : Erster Operand fehlt
094 *RIGHT() : Zweiter Operand fehlt
095 *RIGHT() : Zweiter Operand enthält undefinierte Symbole
096 *RIGHT() : Zweiter Operand muß ein absoluter Ausdruck sein
097 *RIGHT() : Zweiter Operand muß größer als NULL sein
098 *MID() : Erster Operand fehlt
099 *MID() : Zweiter Operand fehlt
100 *MID() : Zweiter Operand enthält undefinierte Symbole
101 *MID() : Zweiter Operand muß ein absoluter Ausdruck sein
102 *MID() : Zweiter Operand muß größer als NULL sein
103 *MID() : Dritter Operand fehlt
104 *MID() : Dritter Operand enthält undefinierte Symbole
105 *MID() : Dritter Operand muß ein absoluter Ausdruck sein

106 *MID() : Startposition befindet sich nicht mehr im String
107 *MID() : Dritter Operand muß größer als NULL sein
108 *UPPER() : Fehlende Argumente

Fehler beim Parsen einer Zeile

112 Unzulässige Zeichen im Labelfeld
113 Lokales Label muß mindestens ein Zeichen lang sein
114 Label darf keine Längenangabe haben
115 Lokales Label muß mit „$" enden
116 Unzulässige Zeichen im Opcodefeld
117 Unzulässige Zeichen am Zeilenende
118 Unzulässige Zeichen im Operandenfeld

Fehler bei der Labeldefinition

120 Illegale Redefinition eines Labels
121 Label kollidiert mit absolutem Symbol
122 Label kollidiert mit importiertem Symbol
123 Label kollidiert mit Registersymbol
124 Label kollidiert mit Makro
125 Label kollidiert mit Registerlistensymbol

Fehler im math. Ausdruck

128 Stacküberlauf - Ausdruck zu komplex
129 Division durch NULL
130 Modulo NULL
131 Addition zweier Label nicht zulässig
132 Verschieben nur um 1...32 Bit(s) möglich
133 Label darf nicht von absolutem Ausdruck subtrahiert werden
134 Importiertes Symbol darf nicht von absolutem Ausdruck subtrahiert werden
135 Operation mit Label nicht zulässig
136 Label aus verschiedenen Sektionen dürfen nicht voneinander subtrahiert werden
137 Klammerfehler
138 Operationszeichen muß „<<" heißen
139 Operationszeichen muß „>>" heißen
140 Unbekanntes Zahlenformat
141 Hexadezimalzahl darf nur 0...9 und A...F enthalten
142 Binärzahl darf nur 0 und 1 enthalten

143 Oktalzahl darf nur 0...7 enthalten
144 ASCII-Zahl darf maximal 4stellig sein
145 Illegale Vorwärtsreferenz
146 Symbol nicht definiert
147 Hexadezimalzahl überschreitet den 32-Bit-Bereich
148 Binärzahl überschreitet den 32-Bit-Bereich
149 Dezimalzahl überschreitet den 32-Bit-Bereich
150 Oktalzahl überschreitet den 32-Bit-Bereich
151 Makro im Ausdruck nicht zulässig
152 Registersymbol im Ausdruck nicht zulässig

Fehler im Addressmode

160 Befehl oder Addressmode erst ab MC68010 erlaubt
161 Befehl oder Addressmode erst ab MC68020 erlaubt
162 Befehl nur für FPU MC 68881/2 zulässig
163 Unmittelbare Konstante vom falschen Typ
164 Base-Displacement muß absolut,relativ oder importiert sein
165 Outer-Displacement muß absolut,relativ oder importiert sein
166 Addressmode enthält Syntaxfehler
167 Unzulässiges Basisregister (nur An oder PC)
168 Unzulässiges Indexregister (nur An oder Dn)
169 „PC“ als Indexregister nicht zulässig
170 Längenangabe muß .W oder .L sein
171 Scalar muß ein absoluter Ausdruck sein
172 Scalar darf nicht NULL sein
173 Scalar darf nur 1,2,4 oder 8 sein
174 16-Bit-Distanz muß absolut oder importiert sein
175 Distanz überschreitet vorzeichenbehafteten 16-Bit-Bereich
176 8-Bit-Distanz muß absolut oder importiert sein
177 Distanz überschreitet vorzeichenbehafteten 8-Bit-Bereich
178 PC-relativ adressiertes Ziel ist weiter als +/-32768 Bytes entfernt
179 PC-relativ adressiertes Ziel ist weiter als +/-128 Bytes entfernt
180 Komma erwartet
181 „)“ erwartet
182 „:“ erwartet
183 „{„ erwartet
184 „}“ erwartet
185 Bitfeld-Offset muß ein absoluter Ausdruck sein
186 Bitfeld-Offset muß 0...31 sein
187 Bitfeld-Offset muß „Dn“ sein
188 Bitfeld-Width muß ein absoluter Ausdruck sein

189 Bitfeld-Width muß 0...31 sein
190 Bitfeld-Width muß „Dn“ sein
191 Byteoperation mit Adreßregister nicht zulässig
192 Operandenlänge .L nicht zulässig
193 Operandenlänge .B nicht zulässig
194 Operandenlänge muß .B sein (kann weggelassen werden)
195 Operandenlänge muß .W sein (kann weggelassen werden)
196 Operandenlänge muß .L sein (kann weggelassen werden)
197 Addressmode für diesen Befehl nicht zulässig
198 Spezialregister <Reg> nicht zulässig
199 Registerliste nicht erlaubt
200 Registerliste erwartet
201 Register erwartet
202 Addressmode „Dn“ nicht erlaubt
203 Addressmode „An“ nicht erlaubt
204 Addressmode „(An)“ nicht erlaubt
205 Addressmode „(An)+“ nicht erlaubt
206 Addressmode „-(An)“ nicht erlaubt
207 Addressmode „(bd,An)“ nicht erlaubt
208 Addressmode „(bd,An,Rn)“ nicht erlaubt
209 Addressmode „($xxx).w“ nicht erlaubt
210 Addressmode „($xxx).l“ nicht erlaubt
211 Addressmode „(bd,PC)“ nicht erlaubt
212 Addressmode „(bd,PC,Rn)“ nicht erlaubt
213 Addressmode „#k“ nicht erlaubt
214 PC-relativ adressiertes Ziel liegt in einer anderen Sektion
215 FAIL <Zeile>
216 Blockgröße darf nicht negativ sein
217 Befehl oder Addressmode nur für PMMU 68030/68851 erlaubt
218 Befehl oder Addressmode nur für PMMU 68030 erlaubt
219 Befehl oder Addressmode nur für PMMU 68851 erlaubt
220 Befehl oder Addressmode nur für Copper erlaubt

Fehler im ersten Befehlsoperanden

224 Erster Operand muß „Dn“ sein
225 Erster Operand muß „Dn“ oder „An“ sein
226 Erster Operand muß „Dn“ oder „-(An)“ sein
227 Erster Operand muß „Dn“ oder „(d16,An)“ sein
228 Erster Operand muß „Dn“ oder „#k“ sein
229 Erster Operand muß „Dn:Dn“ sein
230 Erster Operand muß „An“ sein

231 Erster Operand muß „-(An)“ sein
232 Erster Operand muß „#k“ sein
233 Erster Operand muß absolut sein
234 Erster Operand muß 0...7 sein
235 Erster Operand muß 1...8 sein
236 Erster Operand muß 0...15 sein

Fehler im zweiten Befehls-Operanden

240 Zweiter Operand muß „Dn“ sein
241 Zweiter Operand muß „Dn“ oder „An“ sein
242 Zweiter Operand muß „Dn“ oder „-(An)“ sein
243 Zweiter Operand muß „Dn:Dn“ sein
244 Zweiter Operand muß „An“ sein
245 Zweiter Operand muß „(An)+“ sein
246 Zweiter Operand muß „-(An)“ sein
247 Zweiter Operand muß „(d16,An)“ sein
248 Zweiter Operand muß „#k“ sein
249 Zweiter Operand muß „CCR“ oder „SR“ sein
250 Zweiter Operand muß Spezialregister sein
251 Zweiter Operand überschreitet den 8-Bit-Bereich
252 Zweiter Operand muß absolut sein
253 Zweiter Operand muß 0...15 sein

Fehler im dritten Befehls-Operanden

256 Dritter Operand muß „(Rn):(Rn)“ sein
257 Dritter Operand muß „#k“ sein
258 Dritter Operand muß absolut sein
259 Dritter Operand muß 0...7 sein

Fehler in den IF-Konstruktionen

264 Falsche IF/ENDIF-Paare
265 Falsche IF/ENDIF-Paare in Include-Datei
266 ENDIF ohne zugehöriges IF
267 ELSE ohne zugehöriges IF/ENDIF-Paar
268 Mehrfaches ELSE zwischen einem IF/ENDIF-Paar
269 Argument von IFD/IFND muß ein Symbol sein
270 Erstes Argument von IFC/IFNC fehlt
271 Zweites Argument von IFC/IFNC fehlt

272 Ausdruck nach IFcc enthält undefinierte Symbole
273 Ausdruck nach IFcc muß absolut sein

Spezielle Fehler bei Branch-Befehlen

280 Branch-Ziel muß ein Label sein
281 Branch-Ziel muß in der gleichen Sektion liegen
282 Branch-Ziel weiter als +/-32768 Bytes entfernt
283 Branch-Ziel weiter als +/-128 Bytes entfernt

Spezielle Fehler bei EQU

288 Symbolname vor EQU fehlt
289 Symbol kollidiert mit Label
290 Illegale Redefinition eines Symboles
291 Symbol kollidiert mit importiertem Symbol
292 Symbol kollidiert mit Registersymbol
293 Symbol kollidiert mit Makro
294 Symbol kollidiert mit Registerlistensymbol

Spezielle Fehler bei EQUR

296 Symbolname vor EQUR fehlt
297 Registersymbol kollidiert mit Label
298 Registersymbol kollidiert mit absolutem Symbol
299 Registersymbol kollidiert mit importiertem Symbol
300 Registersymbol kollidiert mit Makro
301 Registersymbol kollifiert mit Registerlistensymbol
302 Ausdruck nach EQUR muß ein Register sein

Spezielle Fehler bei REG

304 Symbolname vor REG fehlt
305 Registerlistensymbol kollidiert mit Label
306 Registerlistensymbol kollidiert mit absolutem Symbol
307 Registerlistensymbol kollidiert mit importiertem Symbol
308 Registerlistensymbol kollidiert mit Registersymbol
309 Registerlistensymbol kollidiert mit Makro

Spezielle Fehler bei SET

312 Symbolname vor SET fehlt
313 Ausdruck nach SET enthält undefinierte Symbole

Spezielle Fehler bei CNOP/ALIGN

320 Operand von CNOP/ALIGN enthält undefinierte Symbole
321 Alignment muß 1...4096 sein

Spezielle Fehler bei XDEF

328 Argument von XDEF muß ein Symbol sein
329 Symbol ist bereits exportiert
330 Ein importiertes Symbol darf nicht exportiert werden
331 Ein Registersymbol kann nicht exportiert werden
332 Ein Makro kann nicht exportiert werden
333 Ein Registerlistensymbol kann nicht exportiert werden

Spezielle Fehler bei XREF

336 Nur globale Symbole dürfen exportiert werden
337 Importiertes Symbol kollidiert mit Label
338 Importiertes Symbol kollidiert mit absolutem Symbol
339 Symbol ist bereits importiert
340 Importiertes Symbol kollidiert mit Registersymbol
341 Importiertes Symbol kollidiert mit Makro
342 Importiertes Symbol kollidiert mit Registerlistensymbol
343 XREF/XDEF darf nur in zu linkenden Programmen vorkommen

Spezielle Fehler bei DC.x

352 Bytekonstante muß absolut oder importiert sein
353 Wortkonstante muß absolut oder importiert sein
354 Langwortkonstante muß relativ,absolut oder importiert sein
355 Bytekonstante muß -128...255 sein
356 Wortkonstante muß -32768...65535 sein

Spezielle Fehler bei RS.x

360 Symbolname vor RS.x fehlt
361 Ausdruck nach RS.x enthält undefinierte Symbole

362 Ausdruck nach RS.x muß absolut sein
363 Ausdruck nach RSSET enthält undefinierte Symbole
364 Ausdruck nach RSSET muß absolut sein

Spezielle Fehler bei INCBIN

368 INCBIN : Längenangabe enthält undefinierte Symbole
369 INCBIN : Längenangabe muß ein absoluter Ausdruck sein
370 INCBIN : Längenangabe muß größer als NULL sein

Spezielle Fehler bei BASEREG

376 Unbekannte BASEREG-Option
377 Basisadresse muß ein Label sein

Allgemeine Fehler

384 Fataler Fehler - Kein Speicher mehr
385 Unzulässige Angabe der Operandenlänge
386 Unzulässige Operanden
387 Fehlende Operanden
388 Direktive <%s> nicht implementiert
389 String endet nicht mit „<Hochkomma>“
390 Dateiname nach INCLUDE fehlt
391 Code-Phase-Fehler : Letzter Pass xxx Bytes, jetziger Pass xxx Bytes
392 PC seit dem letzten Pass verändert
393 Assembler abgebrochen ...

Interne Assembler-Fehler

400 #01
401 #02

Diese Fehler weisen auf einen internen Assemblerfehler hin. Bitte melden Sie diesen Fehler und eine genaue Beschreibung des Programmes, bei dem er auftrat, damit er möglichst schnell behoben werden kann.

Fehler bei FPU-Operationen

Fehler im mathematischen Ausdruck
408 Fließkommazahlenüberlauf
409 Exponent für Single-Zahl zu groß
410 Exponent erwartet
411 Fließkommazahl erwartet
412 FPU-Format Packed decimal nur mit FPU verfügbar
413 mathieeedoubbas- und mathieeedoubtrans.library fehlen

Fehler im Addressmode

416 Operandenlänge nur für FPU-Befehle zulässig
417 FPU-Register im Operanden nicht zulässig
418 Einregistersyntax für diesen Befehl nicht zulässig
419 Operandenlänge muß .X sein (kann weggelassen werden)
420 Quelloperand Dn nur bei Single-Datentypen (.B,.W,.L,.S) zulässig
421 Statischer K-Faktor {#k} muß ein absoluter Ausdruck sein
422 Statischer K-Faktor {#k} muß -64...63 sein
423 K-Faktor muß „{Dn}“ oder „{#k}“ sein
424 Kontrollregister muß „FPCR“,“FPSR“ oder „FPIAR“ sein
425 Addressmode „An“ nur für FPIAR zulässig
426 ROM-Offset #k muß ein absoluter Ausdruck sein
427 Unzulässiger ROM-Offset - Siehe MC 68881 Dokumentation
428 Operandenlänge muß .L oder .X heißen
429 Registerliste darf nur FPU-Kontrollregister enthalten
430 Registerliste darf nur FPU-Datenregister enthalten
431 Registerliste darf nur aus einem Register bestehen
432 FPn nach „:“ erwartet
Fehler im ersten Operanden
448 Erster Operand muß „FPn“ sein

Fehler im zweiten Befehlsoperanden

464 Zweiter Operand muß „FPn“ sein

Fehler im Addressmode

472 MMU-Register im Operanden nicht zulässig
473 Addressmode „Dn“ oder „An“ für DRP,SRP oder CRP nicht zulässig
474 PCSR darf nur gelesen werden
475 MMUSR bei PMOVEFD unzulässig

476 Erster Operand bei FMOVEFD darf kein Kontrollregister sein
477 Vierter Operand muß An sein

Fehler im ersten Operanden

488 Erster Operand muß „#k", „Dn", „SFC" oder „DFC" sein
489 Erster Operand muß „An" oder „VAL" sein

Fehler im zweiten Operanden

496 Zweiter Operand muß MMU-Kontrollregister sein

Fehler bei Copper-Operationen

504 Copper-Zielregisteroffset ungerade
505 Y-Position muß 0...255 sein
506 X-Position muß 0...511 sein
507 Y-Maske muß ein absoluter Ausdruck sein
508 X-Maske muß ein absoluter Ausdruck sein

Anhang D: Register

Name	68000	68010	68020	68030	68881	68851
CRP						*
SRP						*
DRP						*
TC						*
PCSR						*
PSR/MMUSR				*		*
CAL						*
VAL						*
SCC						*
AC						*
BAC0 ... BAC7						*
BAD0 ... BAD7						*
TT0 ... TT1				*		
FPIAR					*	
FPSR					*	
FPCR					*	
FP0 ... FP7					*	
ISP			*	*		
MSP			*	*		
VBR		*	*	*		
SFC		*	*	*		
DFC		*	*	*		
CACR			*	*		
CAAR			*	*		
USP	*	*	*	*		
CCR	*	*	*	*		
SR	*	*	*	*		
PC	*	*	*	*		
SP	*	*	*	*		
A0 ... A7	*	*	*	*		
D0 ... D7	*	*	*	*		

Anhang E, Befehle

Name	68000	68010	68020	68030	68881	68851	Copper
ABCD	*	*	*	*			
ADD	*	*	*	*			
ADDA	*	*	*	*			
ADDI	*	*	*	*			
ADDQ	*	*	*	*			
ADDX	*	*	*	*			
AND	*	*	*	*			
ANDI	*	*	*	*			
ASL	*	*	*	*			
ASR	*	*	*	*			
Bcc	*	*	*	*			
BCHG	*	*	*	*			
BCLR	*	*	*	*			
BFCHG			*	*			
BFCLR			*	*			
BFEXTS			*	*			
BFEXTU			*	*			
BFFFO			*	*			
BFINS			*	*			
BFSET			*	*			
BFTST			*	*			
BKPT			*	*			
BSET	*	*	*	*			
BTST	*	*	*	*			
CALLM			*				
CAS			*	*			
CAS2			*	*			
CHK	*	*	*	*			
CHK2			*	*			
CLR	*	*	*	*			
CMP	*	*	*	*			
CMP2			*	*			
CMPA	*	*	*	*			
CMPI	*	*	*	*			
CMPM	*	*	*	*			
CEND							*

Name	68000	68010	68020	68030	68881	68851	Copper
CLMOVE							*
CMOVE							*
CSKIP							*
CWAIT							*
DBcc	*	*	*	*			
DIVS	*	*	*	*			
DIVS.L			*	*			
DIVSL			*	*			
DIVU	*	*	*	*			
DIVU.L			*	*			
DIVUL			*	*			
EOR	*	*	*	*			
EORI	*	*	*	*			
EXG	*	*	*	*			
EXT	*	*	*	*			
EXTB			*	*			
FABS					*		
FACOS					*		
FADD					*		
FASIN					*		
FATAN					*		
FATANH					*		
FBcc					*		
FCMP					*		
FCOS					*		
FCOSH					*		
FDBcc					*		
FDIV					*		
FETOX					*		
FETOXM1					*		
FGETEXP					*		
FGETMAN					*		
FINT					*		
FINTRZ					*		
FLOG10					*		
FLOG2					*		
FLOGN					*		
FLOGNP1					*		
FMOD					*		
FMOVE					*		

Name	68000	68010	68020	68030	68881	68851	Copper
FMOVECR					*		
FMOVEM					*		
FMUL					*		
FNEG					*		
FNOP					*		
FREM					*		
FRESTORE					*		
FSAVE					*		
FSCALE					*		
FScc					*		
FSGLDIV					*		
FSGLMUL					*		
FSIN					*		
FSINCOS					*		
FSINH					*		
FSQRT					*		
FSUB					*		
FTAN					*		
FTANH					*		
FTENTOX					*		
FTRAPcc					*		
FTST					*		
FTWOTOX					*		
ILLEGAL	*	*	*	*			
JMP	*	*	*	*			
JSR	*	*	*	*			
LEA	*	*	*	*			
LINK	*	*	*	*			
LSL	*	*	*	*			
LSR	*	*	*	*			
MOVE	*	*	*	*			
MOVEA	*	*	*	*			
MOVEC		*	*	*			
MOVEM	*	*	*	*			
MOVEP	*	*	*	*			
MOVES		*	*	*			
MULS	*	*	*	*			
MULS.L			*	*			
MULU	*	*	*	*			
MULU.L			*	*			

Name	68000	68010	68020	68030	68881	68851	Copper
NBCD	*	*	*	*			
NEG	*	*	*	*			
NEGX	*	*	*	*			
NOP	*	*	*	*			
NOT	*	*	*	*			
OR	*	*	*	*			
ORI	*	*	*	*			
PACK			*	*			
PBcc						*	
PDBcc						*	
PFLUSH				*		*	
PFLUSHA				*		*	
PFLUSHS						*	
PFLUSHR						*	
PLOADR				*		*	
PLOADW				*		*	
PMOVE				*		*	
PMOVEFD				*		*	
PRESTORE						*	
PSAVE						*	
PScc						*	
PTESTR				*		*	
PTESTW				*		*	
PTRAPcc						*	
PVALID						*	
RESET	*	*	*	*			
ROL	*	*	*	*			
ROR	*	*	*	*			
ROXL	*	*	*	*			
ROXR	*	*	*	*			
RTD		*	*	*			
RTE	*	*	*	*			
RTM			*				
RTR	*	*	*	*			
RTS	*	*	*	*			
SBCD	*	*	*	*			
Scc	*	*	*	*			
STOP	*	*	*	*			
SUB	*	*	*	*			
SUBA	*	*	*	*			

Name	68000	68010	68020	68030	68881	68851	Copper
SUBI	*	*	*	*			
SUBQ	*	*	*	*			
SUBX	*	*	*	*			
SWAP	*	*	*	*			
TAS	*	*	*	*			
TRAP	*	*	*	*			
TRAPcc			*	*			
TST	*	*	*	*			
UNLK	*	*	*	*			
UNPK			*	*			

Index

! 220
!= 220
» ..... 101, 22, , 437, 456
«= 223, 226
#endif 164, 471
#error 164
#if 164, 471
#include 163, 259
#line 163
#pragma 259, 473
#pragma amicall 182, 301, 476
#pragma header 157, 185
% 226
%= 223, 226
& 221, 226
&& 220, 226
&= 223, 226
* 453
*= 223, 228
+ 218, 437
+= 223, 228
, ... 224
- 218, 453
-= 223
-g20 158
-g30 158
3.0-Standard 509
< 220
<< 218
<= 220
= 223, 22, 256, 441
== 220
?: 222
^ 221, 226
^= 223, 226
_MOVEMBYTES 739
_wbmain 158
__COMPMODE__ 481
__cplusplus 481
__DATE2__ 157
__DATE__ 480
__FILE__ 480
__LINE__ 480
__MAXON__ 157, 481
__MEMFLAGS 604
__RS 739, 754
__STDC__ 481
__TIME__ 480
| 101, 22, 226

|= 223, 226
|| 221, 226
~ 174, 221

A

Abgeleitete Klassen 366
Abhängigkeiten 49
Abort 534
Abrufen der Projektübersicht 692
Abs 529
Absoluter Code 750
Abspielen 105
Abstrakte Klassen 381
Acos 564
Addition 218
Adressierungsarten 735
ALIGNment 747
Amiga-Betriebssystem 300
AmigaGuide 681
AmigaGuide-Projekte 716
Anhalten 105
ANSI C++ 509
ANSI-Modus 258
Antiquitäten 337

Anweisungen 230
- break 240
- case 242
- continue 240
- do 237
- for 236
- if 230
- return 244
- switch 242
- Throw 512
- while 235

Anwendungsgebiete für Onlineprojekte 700
AppIcon 659
Arcsin 512
ARexx 682, 701

ARexx-Befehle 39
- addtx 40
- addtxobj 40
- break 39
- compile 39
- editscript 41
- errors 40
- errorscript 41
- exefile 40
- getinfo 41

info 40
leavemk 40
loadmk 40
newmk 40
objectfile 40
opcomp 40
openmake 40
oplink 40
opsys 40
Options results: 41
quit 39
reopen 39
run 39
saveconf 40
savesession 40
setopt 40
shutdown 39
ARexx-Port 123, 151
Argc 285
Argumemt-Matching 501
Argumente 62, 657
Argumentliste 246, 493
Argv 285
Asctime 586
Asin 564

Assembler 735
Optionen 735
Variablen 738

Assert 556
Atan 564
Atan2 565
Atexit 475, 534
Atof 529
Atoi 528
Atol 528
Auflösung wählen 113
Aufnehmen 105
Aufteilung des Textes 721
Aufzählungen 316
Ausnahme-Deklarationen 521
Ausnahmebehandlung 623
Ausnahmen 512
Ausschneiden 93
Auswertungsregeln 226
Auszeichnen 102
Auto 171
Auto-Indent 103
Auto-Save 114
Automatische Daten 292
Autoren V-2

B

BACKSPACE 87
Basisklassen 396
Bedingungen 230
Benutzerdefinierte Menüs 117
Benutzung der Onlineprojekte 699
Bezeichner 209
Bibliotheken 159
Bit-Manipulationen 219
Bitfelder 332
Blättern im Projekt 691
Block drucken 95
Block einlesen 95

Blockoperationen
- Cut, Copy, Paste 92
- Einrücken 95
- Laden, Speichern, Drucken 94
- Spaltenblöcke 94
- Umwandeln 95

Bookmarks 102
BOOPSI Gadgets 630
Break 240, 474
Breakpoint 664
Breakpoints 664
Bsearch 536
BUFSIZ 551

C

C-Modus 258
Calloc 532
Case 180, 242
Casting 228
Catch 28, 158, 512, 519
Catch-Handler 515
Ceil 565
Char 211
CHAR_BIT 561
CHAR_MAX 561
Class Exception 605
CleanupModules 65, 615
Clearerr 549
CLI 86, 285

CLI-Compiler 69
- Ausgaben 72
- Codeerzeugung 75
- Linker 79
- Optionen 71

Clock 584
CLOCKS_PER_SEC 584
Clock_t 582
Codeerzeugung 158
Compiler is in Panic! 186

Compiler-Fehlermeldungen 99
"##" at end of Macro 162
"#pragma header" may appear on file scope only 185
"(" expected 179
")" expected 162, 166
"," expected 163
"." need a class expression as left operand 174
":" expected 167, 180
" expected 165
"amicall": int/pointer variable expected 182
"amicall": non-overloaded function expected 182
"break" without matching iteration or switch statement 179
"case" requires constant expression 180
"case" without "switch" 180
"catch" expected 184
"continue" used outside iteration statement 179
"default" without "switch" 180
"else" without "if" 179
"Garbage" at end of preprocessor command 163
"id1" defined in classes "id2" and "id3" 176
"inline" used after definition or use of function 171
"inline" used with storage class specifier 171
"sizeof" needs an expression or a type in "()""s 174
"throw": Expression expected 185
"while" expected 179
"]" expected 169
"{" expected 184
"}" expected 169
+= / -= used on pointer: integral expr. as 2nd operand required 179
A global anonymous union must be declared "static" 165
A template declaration may only appear as a global declaration 183
Access to ambiguous base class 169
Access to private base class 172
Access to variable that was constructed under condition 168
Address operator "&" needs a l-value 170
Allocation of vectors: no initializer allowed 176
Ambiguous conversion function 173
Ambiguous function call 171
Ambiguous: use constructor or user conversion? 175
Anonymous union: Illegal member "xxx" 168
Argument missing 171
Arguments do not match function template 184
Assignment: Left value must be L-Value 172
Assignment: Type conflict 170
At least one "#endif" is missing 164
Attempt to assign to const object 178

Attempt to assign to undefined class 179
Attempt to change access of member "xxx" of class "yyy" 168
Automatic variable used outside of its function 182
Binary operator "op" used on illegal arguments 174
Bitfield member has illegal size 169
Calling pure virtual function 177
Can not copy file "source" to "dest" 163
Can not define qualified identifier 167
Can not mix declarations and other statements in C mode 181
Can not open file "name" 163
Can not open header file "name" 185
Can not open math library 164
Can not overload old-style functions 180
Can not overload operator "delete" 176
Can not subtract pointers of different types 178
Can not take address of a "register" object 182
Can not take address of register parameter 181
Can't convert "qualified type*" to plain "type*" 172
Can't reference "const TYPE" with plain "TYPE &" 182
Can't convert virtual base class to derived class 177
Char constant: unmatched 162
Class "id1" is abstract 177
Class members can not get initialized 167
Class name collides with template name 184
Compiler is in Panic! 163, 186
Compiler stack overflow! 164
Condition expected 178
Conditional expression: ":" expected 178
Constant appears twice in same "switch" statement 180
Constant expression as template argument expected 183
Constant integer expression expected 169
Constant String is too long 169
Constructor initializer: "{" expected 177
Conversion "int" - "enum": allowed in C mode only 172
Conversion function not found 173
Conversion functions may not be static 173
Conversion functions may not have an explicit return type 174
Conversion functions must be member of a class 173
Ctor initializer: "(" expected 175
Data type undefined 167
Declaration expected 166
Declaration: can not use a "( )" initializer here 175
Declarations are not allowed here 165
Default Argument not at end 171
Default parameter has illegal type 170
Definition has illegal Type 166
Destructor declaration: Empty parameterlist "()" expected 175
Destructor name does not match class name 175
Destructor: class name expected after "~" 174
Division by zero 178
Double use of parameter register 171
Each "switch" statement may have only one "default" label 180

Empty source! 174
Error in condition 164
Every reference needs an initializer 182
Expression does not match function return type 180
Expression does not match type of "switch" expression 180
Expressions left and right of ":" need common type 178
Filename expected 163
Friend declaration: "xxx" unknown 166
Function "xxx" has internal linkage and is undefined 181
Function already used 182
Function call: can not use register A4 in small data model 170
Function call: Expression left of () is not a function 170
Function declaration: Illegal storage class 171
Function has illegal return type 166
Function needs return value 179
Function parameters differ from first declaration 165
Function redefined 171
Function templates: Argument "name" never used 184
Functions can not be defined local to functions 165
Header file has wrong version number - please recompile it! 185
Header file is invalid! 185
Identifier "name" redefined 166
Identifier "xxx" not defined 169
Identifier expected 162, 167
Illegal argument 171
Illegal jump into try block 185
Illegal parameter for "operator = ()" 178
Illegal pointer-to-member expression 172
Illegal referenced type 166
Illegal type conversion 170
Illegal use of "friend" specifier 168
Illegal use of Bitfields 168
Illegal use of function specifier 171
Illegal use of storage class specifier 167
Implicit conversion "void*" - "TYPE*": allowed in C Mode only 181
Incomplete type-specification 165
Inconsistent exception specification 184
Inconsistent linkage 171
Inconsistent use of register parameters 181
Initialized object is not an aggregate 169
Initializer for Member "name" expected 175
Initializer has illegal Type 169
Integral or enumerated expression expected 180
Internal error in Code generation 185
Jump past initialization of object "name" 179
Label "name" redefined 179
Label "name" undefined 179
Label expected 179
Library vector Offset ("= short constant") expected 181
Line number expected 164
Linkage specification may only occur in file scope 172
Local function definitions can only be "extern" 172

Macro "name" redefined....162
Macro expansion: "(" expected....162
Macro expansion: Illegal use of "#" or "##"....163
Macro name expected....163
Member "name" not initialized....175
Member function call: class expression has illegal type....172
Member function re-declared....166
Member of Class "xxx" expected....169
Member or baseclass initialized twice....175
Memory overflow!....162
Multiple use of same base class....168
Name of baseclass or nonstatic member expected....175
Name of pointer/int variable expected....181
No "#pragma header" found....185
No access to member "name1" of class "name2"....167
No actual "this" pointer....166
No assignment to vectors, functions, and void expressions....178
No default constructor for class "name"....175
No matching "#if"....164
No matching constructor....175
No such member function in class "name"....184
Old-style parameter declaration expected....180
Operator "delete" must return "void"....176
Operator "delete": 1st parameter must have type "void *"....176
Operator "delete": 2nd parameter must have type "size_t"....176
Operator "delete": too many parameters....176
Operator "new" must return a pointer....176
Operator "new": first operand must be of type "size_t"....183
Operator "new": type expected....176
Operator "sizeof": Type has undefined size....174
Operator or type description expected....173
Operator overload: at least one parameter must be "class"....173
Operator overload: Illegal number of formal parameters....173
Operator requires integral operands....178
Operators "++" and "--" need a l-value....170
Operators may not have default arguments or ellipsis....173
Operators must not be static members....174
Overflow at nesting of Includes (max. depth is 25)....163
Parameter has illegal Type....170
Please switch to C-Mode for old style declarations....166
Pointer expression expected....177
Pointer to member: Class expression expected....172
Pointers/references to references are illegal....165
Preprocessor command expected....162
Pseudo-filename expected....164
Recursive definition of macro....163
Redefinition of default argument....171
Redefinition of default argument for parameter "xxx"....171
Reference conversion: L-value expected....177
Referenced type has unknown size....178
Register objects must be local to a function....168
Return type differs form first declaration....167

Second parameter must be "int" 173
Sorry, but ANSI C does not know member functions 181
Sorry, can't convert virtual base class to derived class 181
String constant exceeds line 162
Struct/class "xxx" undefined 166
Subscribt: int expression in []'s expected 170
Subscript: operand before []'s must be vector or pointer 170
Syntax error 172
Syntax error in #pragma 181
Syntax error in base-list 168
Syntax error in declaration 168
Syntax error in enumeration 167
Syntax error in expression or declarator 168
Syntax error in Parameter list 170
Syntax error in type description 166
Syntax error inside class template "xxx" 183
Syntax error inside inlined member function "xxx" 182
Syntax error inside template function "name" 184
Template argument expected 183
Template argument list differs from previous declaration 183
Template argument list: " " expected 183
Template argument list: "," expected 183
Template argument list: "<" expected 183
Template function redefined 184
Template: class or function declaration expected 184
This operator must be a member function 174
This overloaded function has different access modes 166
This template has already been defined 183
This type does not represent an object 176
Too many errors 164
Too many struct/class members 169
Too many Vector elements given 169
Tried to change functions access 167
Type argument expected 183
Type conflict 172
Type conversion by ambiguous user conversion functions 173
Unary operator "op" used on illegal argument 174
Unary operator used on illegal type 170
Unexpected end of source 182
Unions may not be derived 167
Unions may not have members with Constructors or Destructors 165
Unknown function "xxx" 170
Unknown linkage 172
User Error: 164
Vector element type has undefined size 176
Vector has element type of unspecified size 165
Vectors of references, functions or void are illegal 165
Virtual functions can not be static 177
Virtual functions differ in return type 176-177
Virtual functions must be members of a class 177
Void functions may not return data 180
Workspace overflow 186

Compileroption 474
Configuration Icon 114
Const 172, 178, 305, 512
Container-Typen 491
Containerklassen 623
Continue 179, 240
Copper 755, 758
Copyrights und Warenzeichen V-2
Cos 564
Cosh 564
CPPOPT 81
Ctime 586
ctype.h 262, 557

D

Das Vortext-Symbol 691
Datei 114
Datei speichern 193

Dateioperationen
- Laden 92
- Speichern 91

Datentypen 209, 210, 226, 265
- char 211
- int 210
- long 211
- long int 211
- long long int 211
- short int 211
- signed 211
- signed char 212
- unisgned char 212
- unisgned short int 211
- unsigned int 210
- unsigned long int 211
- unsigned long long int 211

Datestamp 45, 582
Debugger 77
Def-Datei laden 123
Default-Argumente 257
Default-Parameter 253
Default: 180
Define 466
Definitionen 251
Deklaration 316
Deklarationen 251, 292
Dekrement 215, 454
DELETE 87, 177, 334, 337, 443, 447
Der Export von Hilfstexten 693

Der HotHelp-Textmarker 720
Destruktoren 354, 393
Die Bedienung von HotHelp 688
Die Hilfsfunktion der HotHelpTools 704
Die integrierte Suchfunktion 695
Die Kapitelübersicht 693
Die Quelldateien 720
Die Schlüssel 722
Die Texte 720
Die Titelzeile des HotHelp-Fensters 689
Die Übersetzung 723
Difftime 585
Direktive 738
Display Beep 115
Div 530
Dividieren 218
Div_t 530
Do 240
DosBase 738
Double 224, 492
Doubletostr 563
Drag&Drop 662
Drucke Zeilennummern 117

E

EASSERT 560
EasyHotHelp 727
EBNF-Darstellung 118
Editor 85

Editor-Fehlermeldungen
(erwartet 198
) erwartet 198
Alles ist rückgängig gemacht worden 190
Alles ist wiederhergestellt worden 190
Altes Pictogramm ersetzen 192
Auflösungen nur im Bereich von 0 bis 2 195
Automode-Bereich nur von 0 bis 2 196
Cursor steht auf keinem Strukturelement 190
Datei existiert bereits 193
endif ohne if 196
Falsches Argument 199
Fehler beim Lesen 194
Funktion würde Einfaltung zerstören 189
Geänderten Text verwerfen 191
IDENTIFIER erwartet 198
Markierung gesetzt 187
Kann Datei nicht öffnen 194
Kann Info-Datei nicht speichern 194
Kann nicht speichern 194
Kein Befehl 198
Kein Block definiert 188

Kein Pictogramm vorhanden 187
Kein Speicher mehr 191, 193
Konnte Backup nicht erzeugen 188
Konnte Datei nicht speichern 194
Konnte Formular nicht öffnen 195
LISToder IDENTIFIER erwartet 199
LVAL erwartet 199
Markierung nicht gesetzt 188
Menü ist zu groß für den Screen 191
NUMBER erwartet 198
Präprocessor-Befehl zu komplex 196
Präprozessor Befeh erwartet 197
Semikolon erwartet 198
STRING erwartet 198
Strukturen stimmen nicht überein 190
Suchmuster nicht gefunden 189
Tabulatorweite nur zwischen 1 bis 20 195
Text ist geschützt 187
TITLE erwartet 197
Wrap-Position darf nicht kleiner als 10 sein 195
Zeichenkete zu lang 189
Zeiten kleiner 30 Sek. machen keinen Sinn! 196
Zu viele Argumente 199
Zu viele Menüeinträge 197
Zu viele Menütitel 197

Editorkommandos 126
Again 137
AutoMode 150
BackSpace 127
BigStepDown 128
BigStepUp 128
BlockHigher 136
BlockLower 136
BlockSL 136
BlockSR 136
Bot 130
Copy 135
Cut 134
Delete 127
Display 149
Do 149
EditForm 148
EndTextRead 155
EnterCmd 148
ExecCli 156
ExecRexx 151
ExecRexxString 151
Exit 146
Find 137
FindForm 147
FindRepForm 147

Fold 139
Font 150
FreeText 131
FreeWin 145
FullSizeWin 146
GetBlockInfo 153
GetLineString 151
GetScreenInfo 154
GetTextInfo 153
GetTextList 152
GetTextPos 152
GetWindowList 152
GetWord 151
GlobalForm 147
GoBookMark 141
GoDispPos 130
GoMatchStruct 138
GoParseError 138
GoPredFold 140
GoPredKey 140
GoStringPos 130
GoSuccFold 140
GoSuccKey 140
GotoForm 147
Iconify 156
IndentReturn 128
InfoForm 148
InitTextRead 154
InsertChars 156
InsertFile 133
Jump 138
KillLine 127
Mark 134
MarkVert 134
ModifyOldCmd 149
MoveDown 129
MoveLeft 129
MoveRight 129
MoveUp 128
New 132
NewText 131
NewWin 145
Open 133
OpenClip 135
OpenDefs 150
PageForm 147
Paste 135
PasteVert 135
PlayRecScript 144
PredText 131
PredWin 145
Print 132

Quit 146
RecPlay 144
RecStart 144
RecStop 144
ReDo 155
RemFold 139
RemTab 138
RepeatCmd 148
Replace 137
ReplaceAll 137
Return 128
Save 133
SaveClip 135
SaveRecScript 144
ScreenDown 129
ScreenUp 129
SelTextN 131
SetBookMark 141
SetFoldsOff 139
SetFoldsOn 139
SetMode 156
SetOverOff 142
SetOverOn 142
SetTab 142
SetWrap 142
SetWrapOff 143
SetWrapOn 142
ShowFold 139
SplitHorWin 146
SplitVerWin 146
SuccText 131
SuccWin 145
TextRead 155
ToggleKey 140
Top 130
UnDo 155
UndoLine 127
UpdateText 133
UpdateTexts 134
WrapPara 143
WrapText 143

Edward 85
Edward.DEF 4
EdwardCommand 119
EFREEMEM 560
Eigenschaften von HotHelp 681
Einfügen 93
Einstellungen 112
Einteilung in Einträge 721
Ellipse 173, 258, 430, 434, 515, 572, 591
Ellipsen-Parameter 434

Else 165, 179
ENDCOP 758
Ende 111
Enforcer 657
ENOMEM 560
ENONUM 560
Entferne Falte 101
Enum 326
EOF 537
EOF 540
ERANGE 529, 559
Errno 559
Ersetzen 98
Erstelle Backup 115
Erzeuge Falte 101
EUSRBRK 559
Exception Handling 509
Exceptions 28
Exit 475, 532, 534, 555
Exp 564
Expo10 567
Exponent 225
Extern 171

F

Fabs 565
Fallunterscheidungen 230
Falten 100
Falten überspringen 117
Farben 113
Fclose 538
Fehler 77
Fehlererkennung 596
Fehlermeldungen 160

Fenster
- Erzeugen 107
- Mit Ansichten arbeiten 109
- Schließen 108
- Wählen 107

Feof 549
Ferror 549
Fflush 551
FFP-Single 3
Fgetc 540
Fgetpos 555
Fgets 541
FILE 536
FILENAME_MAX 539
Fließkommaoperationen 226
Fließkommatypen 224

Float 224-225
Floattostr 563
Floor 565
Fmod 566
Fontgröße 113
Fontname 113
Fopen 538
FOPEN_MAX 539
For 179, 240
Form 591
Form Feed 117
Fpos_t 554
Fprintf 545
Fputc 541
Fputs 542
Fpwr10 567
Fread 553
Free 532
Freopen 539
Frexp 566
Friend 168
Fscanf 548
Fseek 554
Fsetpos 555
Ftell 554
Füllbytes 747

Funktionen 244, 657, 743
- Deklaration 251
- Ellipse 257
- Gültigkeitsbereiche 249
- main 244
- Member-Funktionen 355
- Rekursive 251
- Rückgabe 247
- Überladen 253
- Variable Parameterliste 257
- Vektoren als Parameter 277

Funktions-Templates 498
Funktionsparameter 267
Fuß 116
Fwrite 553

G

Gadgets
- BOOPSI 630
- GadTools 630

GadTools Gadgets 630
Gehe zu Fehler 99
Gehe zur Zeile 99

GetBaseReg 615
Getchar 540
Getenv 535
Gets 541
Getstream 596
Gleitpunktzahl 225
Gmtime 583
Goto 179, 185, 238, 567
Grafiken 723
Großbuchstaben 96
Grundlagen der Projekterstellung 718
Gültigkeitsbereiche 233, 249, 294, 326

H

Hello World 204
Hexadezimal 213
Hilfe über Hilfe 696
HotHelp 680
HotHelpComp - der HotHelp-Übersetzer 728
HotHelpMarker 718
HotHelpPref - HotHelps Voreinstellungen 706
HotHelpPro - Der Projektmanager 714
HotHelpTools 704
HotKey 111, 123, 688
HUGE_VAL 529
Hunks 68
Hypertext 680

I

Iconify 111
IDCMP-Flags 627
IEEE-Single 3
If 165, 179
Implementation 382
Include 470, 754
Information Hiding 358
Initialisieren 343

Initialisierung 336
    von Klassen 393

Initialisierungen 229
InitModules 65, 615
Inkrement 215, 454
Inline 171, 310, 358, 495, 671
Inline-Funktionen 309
Int 210
Intelligente Zeiger 453
IntMult 32 218
Inttostr 563
IOFBF 550

IOLBF 550
IONBF 551
Isalnum 557
Istream 437

J

Jederzeit hilfsbereit 680
Jmp_buf 570
JMP_BUF_SIZE 570

K

Kapitel, Vorgänger und Nachfolger 725
Kernigham & Richie 258
KeyDef 121
Keys 121
Kickstart 658
Klammern Einzug 104

Klasse
- ArgsMCursorC 648
- BufferC 647
- CurrentDirC 646
- DirLockC 646
- EListC 623
- Exception 623
- FileHandleC 647
- FileLockC 646
- GadgetC 630
- GadgetListC 630
- GagdteEventC 630
- GCycleC 630
- gen_array 623
- gen_arraylist 623
- GeometryC 633
- GeometryGroupC 633
- GModelC 630
- GroupLayoutC 632
- GTIDCMPortC 630
- GTWindowC 630
- HandlerPortC 626
- IDCMPortC 629
- InputHandlerC 647
- ios 596
- LayouC 632
- LayouterWindowC 644
- LBBevelboxC 643
- LBBevellineC 644
- LibraryBaseErrC 621
- ListC 623
- MessageHandlerC 626
- OutputHandlerC 647

PortC 625
ResourceX 623
RootGeometryC 639
SignalsC 625
StandardWindowC 619, 645
String 606
TextAttrC 645
WindowC 627
WindowCloseHandlerC 629
WindowEventHandlerC 628
WindowSnapshotC 628

Klassen 353
Klassen-Templates 491
Klassenbibliothek 3
Kleinbuchstaben 96
Kommandozeilen-Interpreter 112
Kommandozeilenargumente 285
Kommandozeilenversion 474
Kommentar 734
Kommentare 54, 204
Konfigurationsdatei 4

Konstante 212
Suffix
l 212
LL 213
U 212
ULL 213

Konstanten 751
Konstruktoren 354, 393-394
Konvertierungsfunktionen 423
Kopf 116
Kopieren 93

L

L-Wert 215-216, 223
Label 738
Labelfeld 734
Labs 529
Laden 117
Laufzeitfehler 517
Ldexp 566
Ldiv_t 530
Lesezeichen 102, 682
Lexikon 680
Linker 59, 79, 158
Linker Rand 117
Linker-Bibliotheken 63
Literale 212, 224
Lizenzbedingungen V-3

LN_SUCC 738
Localtime 584
Log 512, 564
Log10 564
Logfile 60, 63
Long 211
Long double 224
Long int 211
Long long int 211
Longjmp 517, 567

M

MACRO 742
Main 205, 244
Make 13, 43
Makros 105
Malloc 337, 532
Mantisse 225
Markieren 93
Markieren der Texte 720
math.h 263

MaxonSDB
- Menü Einstellungen
  - Einstellung laden 670
  - Einstellung sichern 670
  - Einstellung sichern als 670
- Menü Projekt
  - Debugger beenden 669
  - Öffnen 669
  - Über 669
  - Wieder Starten 669
- Menü Schritt
  - Neue Adresse zeigen 669
  - Programm abbrechen 670
- Programmstart 659
- Vorbereitung 659

MCPPC 69
Mehrfach-Anweisungen 54
Member 420
Member-Funktionen 355, 377, 495
Memchr 582
Memcmp 582
Memcpy 159, 581
Memmove 159, 581
Memset 582
Menu 119
MenuGroup 119
MenuItem 120
Mktime 584
MMU 735

Mnemonics 734
ModeQualList 121
Modf 566
Modul 657
Möglichkeiten, HotHelp aufzurufen 697
Multiplizieren 218

N

NARG 739
Neuer Font 113
New 176, 334, 337, 443-444
Nil: 535
No errors 185
NULL 533, 537, 575
Nummernbereich 117

O

Objekte 353
Offsetof 575
Oktal 213
Onlineprojekte 681, 699, 715
Onlineprojekte und andere Editoren 700
Opcodefeld 734
Operandenfeld 734

Operator
- ! 597
- & 182
- && 597
- || 597
- ## 470
- * 453
- + ... 461
- delete 447
- Funktionsaufruf 450

Operatoren 214, 437
- ++ 454
- += 157
- -- 454
- = ... 441
- ?: 222
- Komma-Operator 224
- Linksshift 219
- Logische
  - && 221
  - || 221
- Multiplikative
  - % 217
  - * 217
  - / 217

Postfix
++ 215
-- 215
Shift
<< 218
>> 219
unäre 229
Unäre
! 216
+ 216
++ 215
- 216
-- 215
sizeof 216
~ 216
Zuweisungsoperatoren 223
|= 157

ORG 750
Ostream 437, 591
Overload 255

P

Parameter 245, 657
Parameterliste 246
Parameterlisten 259
PARAMETER_BASE 562
Perror 555
Pfad 115
Pointer 265
Pow 565

Preprozessor 73, 85, 157, 205, 301, 465, 480
Arbeitsweise 482

Preprozessoranweisung 470
Printf 542
Private 167
Programmausführung 474
Projekt & Ende 110
Projekt- und ARexx-Querverweise 725
Projekttest 716

Projektverwaltung 45
Abhängigkeiten 48
Änderungen per Hand 52
Compileranweisungen 50
Dateien hinzufügen 48
Kommentare 54
Makros 55
Projektpflege 49
Regeln 56

Übersetzen ... 48
Übersetzung erzwingen ... 50
wechseln ... 50
Zeilenverbindung ... 54

Protected ... 374
Prototyp ... 252
Prozessor-Exception ... 225
Ptrdiff_t ... 575
Public ... 167
Putchar ... 542
Puts ... 542
Pwr10 ... 566

Q

Qsort ... 491, 535
Qualifizierungen ... 292
Querverweise ... 690, 722
QuotedString ... 119

R

Raise ... 571
Rand ... 531
Random ... 531
RAND_MAX ... 531
Realloc ... 533
Rechter Rand ... 104, 117
Redo ... 124
Referenzen ... 265, 288
Register ... 168, 171, 182, 301
Registervariablen ... 301
Rekursiv ... 251
Remove ... 552
Rename ... 552
Reservierte Symbole ... 739
Return ... 160, 180
Rewind ... 554
Rollsymbole ... 690
RSRESET ... 754
RSSET ... 754
Rückgabewert ... 205

S

Scanf ... 546
Schleifen ... 235
Schlüssel, Projekte und Muster ... 691
Schlüsselgruppen ... 725

Schlüsselwort ... 207, 492, 519, 743, 749
class ... 492

template 492

Schriftarten 723
Scope2 ..... 33, 239, 326, 360
Scoperegeln 373
Scrollbereich 115
SECTION-Befehl 749
Seitenformat 116
Semaphors 71
Setbuf 551
Setjmp 517, 567
SetMode 121
Setvbuf 550
Set_new_handler 532
Set_terminate 606
Set_unexpected 605
Shellstart 660
Shift 219
Short int 211
Short-Cuts 112
Signed 211
Signed char 212
SIGTERM 571
Sig_atomic_t 572
SIG_DFL 571
SIG_ERR 571
SIG_IGN 571
Sin 564
Sinh 564
Sizeof 174, 216, 575
Size_t 183, 532-533, 537, 575, 602
Speicherklassen 292
Speichern 117
Sprachstandard 157
Sprintf 545
Sprünge 238
Sqrt 564
Srand 531
Sscanf 549
Stackpointer 739

Standardfunktionen 261
- abs 264
- ceil 263
- cos 263
- exit 263
- exp 263
- fabs 263
- floor 263
- Isalpha 262
- islower 262
- isupper 262

labs 264
log 263
printf 262
qsort 314
rand 264
Random 264
sin 263
sqrt 263
tan 263
tolower 263
toupper 263

Standardprojekte 714
Start der HotHelpTools 705
Startup-Code 63, 65
Static 171, 181, 296, 310, 496, 521
Statische Daten 292
Statische Member 364
Status-Anzeige 86
Stderr 537
Stdin 537
Stdio.h 262
stdlib.h 263
Stdout 537
Std__err 536
Std__in 536
Std__out 536
Steuerzeichen 213
Strcat 576
Strchr 578
Strcmp 577
Strcpy 286, 576
Strcspn 579
STREAM_MAXSTRING 541
Strerror 579
Strftime 585
Stricmp 577
string.h 262
Strings 278
Stringvergleiche 281
Strlen 285, 579
Strlwr 578
Strncat 577
Strncmp 577
Strncpy 576
Stroustrup 514
Strpbrk 579
Ströme 597
Strrchr 578
Strspn 578
Strstr 579
Strtod 528

Strtok 580
Strtol 527
Strtoul 527
Strtouvl 528
Strtovl 528
Struct 323
Struktur-Check 98

Struktur
List 623
MinList 623

Strukturen 321
Strukturieren 100
Strukturkontrolle 99
Strukturoffsets 754
Strupr 578
Subklasse 367
Subtraktion 218
Suchen 96-97
Suffix 212
Superklasse 367
Support V-2
Switch 180, 240, 242
Symbole 480
Syntax 312
System 535

T

TAB in Spaces 103
TAB Weite 104
Tan 512, 564

Tastaturbedienung
HotHelp 697

Templateargument 502
Templates 184, 491
Temporäre Objekte 426

Text
Auswahl 89
Drucken 90
Erzeugen 89
Freigeben 90
Löschen 90

Textdraw 115
Throw 28, 158, 185, 519
Time 583
Time_t 582
Titelzeile 86

Tm 583
Tmpfile 553
Tmpnam 552
Tolower 558
Tooltypes 623
Toupper 558
Trigraph 477, 482
Trigraph-Sequenzen 214
Try 28, 158, 519
Try-Blocks 515
Typdefinitionen 304
Typedef 304, 602
Typen 224
Typregeln 371
Typsicheres Linken 297

Typumwandlung 228
casting 228

U

Überladen 253, 429
new 444

Überschreiben 103
Übersetze CRs 104
Übersetzung 471
UCHAR_MAX 562
Uinttostr 563
Undo 124
Undo Konfiguration 114
UndoLine 87
Unexpected 522, 605
Ungetc 540
Unionen 328
Unsigned char 212
Unsigned int 210
Unsigned long int 211
Unsigned long long int 211
Unsigned short int 211
Uvlongtostr 563

V

Variable Parameterlisten 257
Va_arg 573
Va_end 574
Va_list 572
Va_start 573
Vektordeklarationen 275

Vektoren 265, 271-272

dynamisch erzeugt ... 335
Initialisierung ... 286

Vektorkonzept ... 491
Vererbung ... 371
Vererbungslehre ... 366
Vergleiche ... 220

Verknüpfungen
Bitweise ... 221

Verschiedene Ansichten über HotHelp ... 694
Vertauschen ... 98
Vertikal einfügen ... 94
Vertikal markieren ... 94
Verwendung von Lesezeichen ... 696
Verzweigungen ... 230
Vfprintf ... 545
Vielfachverzweigungen ... 242
Virtual ... 171
Virtuelle Member ... 355
Vlabs ... 530
Vlongtostr ... 563
Void ... 160, 166, 181, 205
Void* ... 176
Volatile ... 172, 305, 308, 569

Vordefinierte Symbole
__COMPMODE__ ... 481
__cplusplus ... 481
__DATE__ ... 480
__FILE__ ... 480
__LINE__ ... 480
__MAXON__ ... 481
__STDC__ ... 481
__TIME__ ... 480

Vorzeichen ... 225
Vprintf ... 545
Vsprintf ... 545

W

Warnungen ... 77, 160
Funktion „name“ has no return statement ... 160
Nested comment ... 160
No prototype for function „name“ ... 160
Precompiled header file "name" not found ... 161
Statement has no effect at all ... 160
Suspect "=" in condition ... 161
Temporary object for non-const reference introduced ... 161
This seems to be an old-style parameter list ... 161
Unknown #pragma (maybe misspelled?) ... 161

Unsafe type conversion 161
Variable name never initialized 161
Variavle xxx declared but never used 160
Very long argument for Macro "xxx" - probably missing ")" 162

Wbparse 159
Wchar_t 534, 576
Which_xdigit 558
While 158, 240
Wildcard 97
Word Wrap 103-104
Workbenchargumente 623
Wortsymbol 209

Z

Zahlenformate 740
Zeige Falte 101
Zeige Spaces 103
Zeige TABS 103

Zeiger 265, 272, 420
auf Funktionen 311
Überladene Funktionen 313

Zeigertypen 268
Zeilen bis zum Text 116
Zeilen nach dem Text 116
Zeilen/Seite 116
Zeilen/Text 117
Zeilenverbindung 54
Zeit 115
Zugriffsrechte 358
Zum Anfang 116
Zum Ende 117
Zuweisungs-Operator 441

Maxon C++ 3

Features

Light-Version
- C++-Compiler (AT&T 2.0 Standard)
- ANSI C-Compiler (Aztec- und Lattice-ANSI-kompatibel)
- CLI-Compiler
- HotHelp 3-Reader zum Anzeigen von HotHelp-Projekten

Developer-Version
- C++-Compiler (AT&T 3.0 Standard)
- Templates, Exceptions
- ANSI C-Compiler (wie Light-Version)
- CLI-Compiler
- beinhaltet zusätzlich: MaxonHotHelp 3, MaxonSDB, MaxonEASY-OBJECTS und MaxonASM-CLI

Compiler (C/C++)
- komfortable Projektverwaltung (Make)
- direkter Fehlerstellenansprung
- voll in den Editor integriert
- über ARexx-Schnittstelle steuerbar
- sehr schnelle Turnaroundzeiten
- echte vorkompilierte Include-Dateien (Preincludes)
- Include-Dateien für Kickstart 3.1
- speicherbarer Programmstatus
- Codegenerator für prozessorspezifischen Code 68010/20/30 und 68881/2 FPU

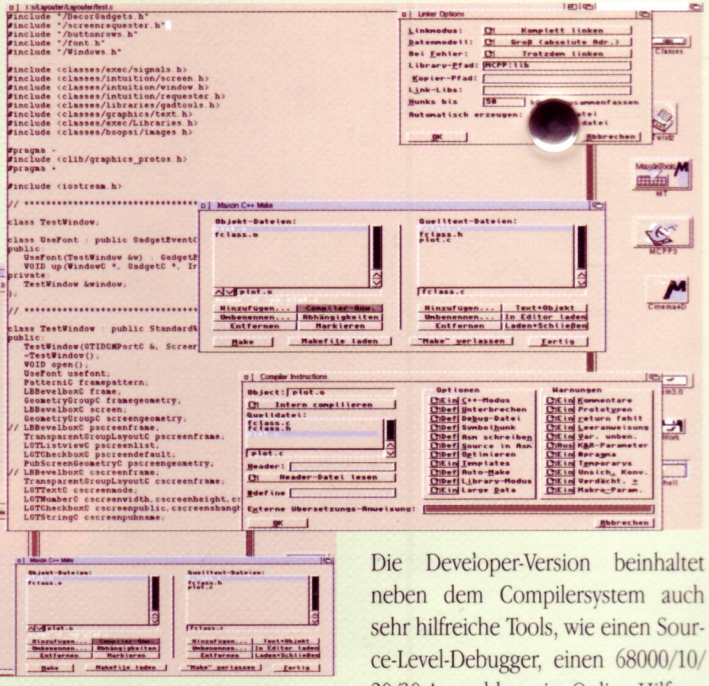
Programmeditor (EDWARD)
- Multi-Window-Technik
- beliebig viele Texte gleichzeitig bearbeitbar (speicherabhängig)
- Tastatur- und Menüfunktionen frei belegbar
- speicherbarer Programmstatus
- ARexx-Schnittstelle

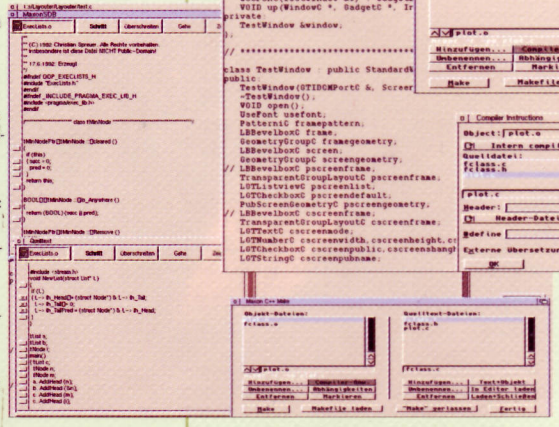
Hilfesystem HotHelp 3
(nur in der Developer-Version)
- Deutschsprachige Beschreibung der Library- und Device-Funktionen zu Amiga-OS 2.1
- jederzeit aktivierbar
- kontextsensitiv
- komfortabler Editor für eigene Projekte
- Online-Projekte (direkte übernahme von Texten und Sourcecode in das Hilfesystem)
- ARexx-Schnittstelle
- Grafikeinbindung direkt im Hilfetext
- direktes Lesen von AmigaGuide™-Projekten
- lizenzfreie Weitergabe des HotHelp 3-Anzeigers

Features der Klassenbibliothek (MaxonEASY-OBJECTS)
(nur in der Developer-Version)
- Ressourcenhandling: Betriebssystemressourcen (Speicher, Signale, Fenster, Bildschirme etc) werden automatisch verwaltet und bei Programmende freigegeben (keine aufwendige Fehlerbehandlung mehr nötig).
- Fehlerbehandlung durch Exceptions
- Handlerkonzept
- Datenstrukturen: Listen, dynamisch große Arrays, Datenpuffer (z.B. für I/O-Operationen)
- Verwaltung folgender Betiebssystemresourcen: Exec: Signale, Nachrichtenports, Speicher, Libraries, Devices, Listen - DOS: Locks, Dateien, Pfadnamen, Kommandozeilenargumente, Notifikation - Utility: Hooks und Taglisten - Intuition: Bildschirme, Fenster, IDCMP-Nachrichten, Systemrequester - Libraries: GadTools, ASL, Icons, Workbench, Tooltypes, Applikationenfenster
- BOOPSI Unterstützung
- automatisch größenveränderbare und font-veränderbare Fenster durch den Layouter
- deutschsprachige Online-Dokumentation

Externer Assembler (MaxonASM-CLI)
(nur in der Developer-Version)
- 68000/10/20/30 Makro-Assembler
- FPU 68881/2 und MMU-Unterstützung
- Multi-Pass-Optimierung erzeugt sehr schnellen und kompakten Programmcode.
- voll in die Projektverwaltung integrierbar

C/C++ Source Level Debugger (MaxonSDB)
(nur in der Developer-Version)
- Multi-Window-Technik
- einfache Bedienung - Drag&Drop
- Beobachten von Variableninhalten
- Haltepunkte
- direkt vom Compiler aus aufrufbar





MaxonC++ ist die erste konsequente Implementierung des C++ 3 Standards auf dem Amiga.

Das MaxonC++-Compilersystem beinhaltet sowohl einen ANSI C-, als auch einen zum AT&T 3.0-Standard kompatiblen C++-Compiler. Der Compiler erzeugt aus den C++-Quelltexten direkt ausführbare Programme ohne Umweg über C, Assembler oder andere Sprachen. Durch die volle Systemintegration werden überzeugend schnelle Turnaroundzeiten erreicht.

Die Developer-Version beinhaltet neben dem Compilersystem auch sehr hilfreiche Tools, wie einen Source-Level-Debugger, einen 68000/10/20/30-Assembler, ein Online-Hilfesystem mit der deutschsprachigen Beschreibung der Systemfunktionen und eine umfangreiche Klassenbibliothek.

Das auch in der Light-Version aufwendige und gut strukturierte deutschsprachige Handbuch ist ein kompletter ANSI-C/C++-Programmierlehrgang und auch für Einsteiger zu empfehlen.

Lauffähig auf allen Amigas mit min. 2 MB Speicher und Amiga-OS 2.0. Bei Nutzung der Klassenbibliothek werden 6 MB Speicher benötigt.

AMIGA®

4 010509 411302

MAXON Computer GmbH • Industriestr. 26
D-65760 Eschborn • Telefon 06196/481811
Irrtümer und Änderungen vorbehalten!

MAXON computer